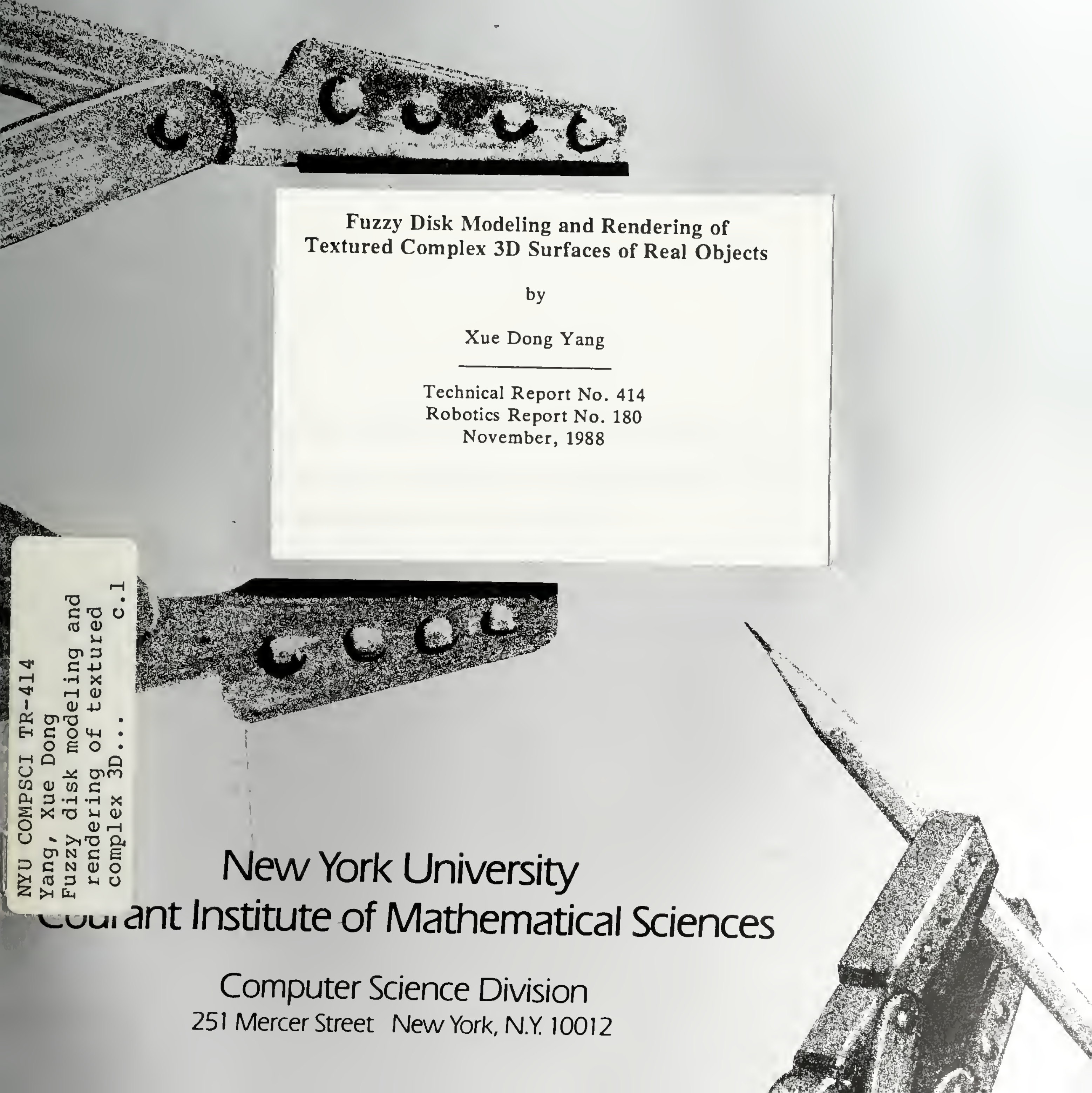
Robotics Research Technical Report



Fuzzy Disk Modeling and Rendering of Textured Complex 3D Surfaces of Real Objects

by

Xue Dong Yang

Technical Report No. 414
Robotics Report No. 180
November, 1988

New York University
Courant Institute of Mathematical Sciences

Computer Science Division
251 Mercer Street New York, N.Y. 10012

Fuzzy Disk Modeling and Rendering of Textured Complex 3D Surfaces of Real Objects

by

Xue Dong Yang

Technical Report No. 414
Robotics Report No. 180
November, 1988

New York University
Dept. of Computer Science
Courant Institute of Mathematical Sciences
251 Mercer Street
New York, New York 10012

Work on this paper has been supported by Office of Naval Research Grant N00014-87-K-0129 National Science Foundation CER Grant DCR-83-20085, National Science Foundation Grant subcontract CMU-406349-55586, and by grants from the Digital Equipment Corporation and the IBM Corporation.

FUZZY DISK MODELING AND RENDERING OF TEXTURED COMPLEX 3D SURFACES OF REAL OBJECTS

by

Xue Dong Yang

September 1988

A dissertation in the Department of Computer Science submitted to the faculty of the Graduate School of Arts and Sciences in partial fulfillment of the requirements for the degree of Doctor of Philosophy at New York University.

Approved: [illegible] [illegible]

Kenneth Perlin

Research Advisor

Jacob T. Schwartz

Research Advisor

ABSTRACT

Three-dimensional geometric modeling in computer graphics is concerned with the representation, specification, and manipulation of free-form curves, surfaces, and volumes. This research explores a model for constructing representations of complex three-dimensional surfaces of real-world objects, such as sculptures in a museum, from sample points acquired with a special 3-D camera, and for synthesizing computer-generated pictures from this model. The difficulty of this problem comes from the complexity of the surface characteristics of such objects, which involve complicated irregular shapes and rich textures.

This thesis presents a new three-dimensional surface model - three-dimensional fuzzy disk model, for computer graphics display. This model allows any curved surface to be approximated by a number of overlapping disks. A new blending method has been developed to generate smoothly curved surfaces from the overlapping disks. The shape of a blending surface can be controlled by varying some geometric parameters. This three-dimensional fuzzy disk representation is organized into a multi-resolution structure which allows adaptive refinement of surfaces details and supports a coarse-to-fine display process.

A scan-line rendering algorithm has been developed to synthesize images from the new model. We also present a simpler, less accurate, but more efficient approximation to the original model. In addition, we present a fast shadow penumbra approximation algorithm capable of generating soft shadows.

ACKNOWLEDGEMENTS

My most sincere thanks go to my research advisors, Professor Ken H. Perlin and Professor Jacob T. Schwartz. Their enthusiasm and encouragement provided invaluable support to my efforts.

The early part of my research projects at the Robotics Research Laboratory of Courant Institute is in computer vision. Professor Jacob Schwartz introduced me to do scientific research and guided me throughout those research projects. The results of that research have been published elsewhere.

This thesis summerizes the later part of my research in computer graphics at Courant Institute. Many ideas in this thesis were born in discussions with Professor Ken Perlin. He provided constant help both with technical problems and expository difficulties. I am very grateful to Professor Jacob Schwartz, Professor Ken Perlin, and Professor Jim Demmel who read the whole thesis and gave many suggestions which improved the presentation and enhanced the clarity of this work. I would also like to thank Professor Paul Wright and Dr. Haim Wolfson for serving as committee members. I would also like to thank Mr. Alan Kalvin who read the whole thesis and made many small corrections.

I would also like to express my gratefulness to Dr. Marc Bastuscheck, Professor Robert Hummel, Professor Alan Siegel, Professor H. Stone for various kind of help and encouragement from them during my years at Courant Institute. Special thanks to Dayton Clark, Jim Fehlinger, Fred Hanson, and Shidan Tavana who provided many technical supports to my research.

Finally, my wife Jain Hua, my lovely daughter Si Si, and my parents, without their supports and encouragements, my accomplishments would not be possible.

Xue Dong Yang

September, 1988

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This research explores a model for constructing representations of textured complex three-dimensional surfaces of real-world objects from sample points acquired with a special 3-D camera, and for synthesizing computer-generated images from this model.

1.1. Three-dimensional Surface Modeling

The three-dimensional geometric modeling problem in computer graphics is concerned with the representation, specification, and manipulation of free-form curves, surfaces, and volumes. The research reported here deals with the surface modeling problem only. In general, a geometric model is only an approximation to a real object. Many real-world objects can not be defined exactly by combinations of mathematical surfaces. In typical geometric modeling, we pick out a set of points on the object and require either that the representation be exact only at those points, or that the distances between the points and the representation be small.

Surface modeling approaches can be divided into two types: surface interpolation and surface fitting. In surface interpolation, a number of surface pieces in a piecewise representation are constructed from the given data points to approximate a surface. When the amount of data is very large, it is desirable to use a more compact representation of the surface to save storage space. Surface fitting is the process of finding a surface defined by a smaller number of data points to approximate a large set of data points.

A mathematical description of three-dimensional surfaces can be either explicit or implicit. In explicit representation, surface patches, such as polygonal and parametric surface patches, are defined explicitly. In implicit representations, an implicit surface is defined as the set of all points which satisfy some equation $S(x,y,z) = 0$; for example, $S(x,y,z)$ may be a quadric or other algebraic function. When a function is non-linear, solutions to it are usually found by numerical methods. But the techniques are applicable to idealized mathematical surfaces only.

1.2. Rendering Process

Any computer rendering process begins with some view-independent three dimensional description of the shapes of objects within the scene. The main computations of such a rendering process typically consist of the following steps:

(1) **Visible Surface Calculation** - Transform the 3D model into an image of the scene as seen from a given view point. Along each viewing ray through a particular pixel of the image, identify the nearest, and therefore visible, surface.

(2) **Surface Texture Calculation** - Compute the texture at each pixel. This will determine how the surface point will respond to illumination.

(3) **Surface Shading Calculation** - Illuminate the surface to get a final color intensity value at the pixel.

The above processes may be either interleaved or performed in sequence.

1.3. The Research of This Thesis

Much previous research has been concerned with the problem of modeling the complex textured three-dimensional surfaces of real-world objects, such as sculptures in a museum. The difficulty of this problem comes from the complexity of the surface characteristics of such objects, which involve complicated irregular shapes and rich textures.

Our research describes a new representation of three-dimensional surfaces. It achieves the following goals:

(1) It defines a new representation from which we are able to generate reasonably accurate, plausibly smoothed curved surfaces for arbitrary real-world objects.

(2) Our method puts the primitives required for representation of real-world objects into a simple uniform format.

(3) The new representation is more efficient overall than existing methods in terms of data representation and image regeneration.

A key motivation for this research is the desire to combine the simplicity of polygonal representation, the beauty of spline surfaces, and the efficiency of octree hierarchical structure. (These three earlier models are reviewed in the next Chapter.)

Approach

(1) The new representation consists of a set of simple surface patches, each a three-dimensional oriented disc, in a uniform format.

(2) Using this model, a reasonably smoothed surface can be generated by blending overlapping surface patches. This is analogous to the traditional way of generating spline surfaces.

(3) The primitive we use is simpler than that of other existing 3D surface patch representations.

(4) Our representation can be organized in a multi-resolution structure for adaptive surface detail refinement and data compression, in a manner analogous to octree hierarchical structure, but no logical (topological) connection between different levels is required in our representation.

Other features of this representation allow efficient rendering, fast hidden-surface removal, adaptive surface detail refinement , etc., in image synthesis.

Experiments reported below use sampled data from real world three-dimensional objects by using video-based 3D sensing techniques. These results allow us to demonstrate characteristics of the new model experimentally, and to indicate applications.

1.4. Overview of Thesis Structure

We begin by reviewing related work in 3D surface modeling.

The new 3D surface representation is then introduced. A process to construct this surface representation from sample data is developed. Characteristics of this model are then discussed.

We also discuss algorithms for synthesizing computer-generated images from our representation. These utilize various properties of the new representation efficiently.

Note that since the new representation we develop is mainly intended for approximating textured complex 3D surfaces of real objects (though it is not restricted to this class), a set of sample points from the surface to be modeled must be provided to construct the surface representation. although 3D sensing techniques are not a central issue for this thesis, we review 3D sensing techniques, for the sake of completeness, particularly those used in our experiments in Appendix A.

Finally, future research directions are discussed.

CHAPTER 2

PRIOR AND RELATED WORK

Major methods for representing three-dimensional objects include surface modeling, solid modeling, and volume density techniques. Surface modeling uses various types of closed surfaces, such as polygonal meshes or parametric patches, etc. to approximate an object's surface. Solid modeling can use any of several schemes: constructive solid geometry (CSG), boundary representation (BR), sweeping, volumetrics, volume decomposition, and primitive instancing. The volume density approach defines a surface as the set of all points in a volume at which a density function satisfies some specified condition.

Among these models, constructive solid geometry, boundary representation, sweeping and primitive instancing are generally not suitable for representing arbitrarily shaped three-dimensional real-world objects. For this reason we shall review only the polygon mesh, parametric patch, volumetric, volume decomposition (octree), and volume density methods. Besides these, we review another interesting related technique, the so called particle system technique, for modeling such fuzzy objects as fire, clouds, and water.

2.1. Polygon Meshes

In general, surfaces of complex three-dimensional objects are too complicated to be described by single equations, and therefore a need for piecewise approximation arises. The simpliest possible approximations are piecewise linear, and involve rectangular or triangular faces.

The curved surface of any 3D object can be approximated by a sufficiently detailed polygon mesh. The errors introduced by polygonal approximation can be made arbitrarily small by using more and more polygons to create a better piecewise approximation.

A variety of detailed schemes for representing polygon meshes is available. Any such mesh is a collection of vertices, edges, and polygons, always involving vertices connected by edges. Three commonly used representations are as follows:

• *Explicit Polygons*. Each polygon is represented by a list of vertex coordinates:

$$P=((x_1,y_1,z_1),(x_2,y_2,z_2),\ \ldots,\ (x_n,y_n,z_n)).$$

• *Polygons Defined by Pointers into a Vertex List*. Each vertex in the polygon mesh is stored just once. A polygon is defined by a list of pointers (or indices) into the vertex list.

• *Explicit Edges*. In this representation we have a vertex list and a edge list. Each edge in the edge list points to the two vertices in the vertex list defining the edge, and also to the one or more polygons to which the edge belongs. A polygon is defined by the list of its edges:

$$P=(E_1,E_2,\ \ldots,E_n).$$

None of these representation makes it easy to determine which edges are incident on a vertex. *Baumgart*[11] (1975) expands the edge description to include pointers to the two adjoining edges of each polygon, while the vertex description includes a pointer to an (arbitrary) edge incident on the vertex, and thus more polygon and vertex information is available.

Generally, in representing arbitrarily shaped real-world objects the number of polygons necessary for a reasonable approximation becomes excessive. This increases both space requirements and the execution time of algorithms which process the representation. Authors have therefore attempted to consider higher order surfaces, and curved piecewise surface patches - the parametric surface patch approach discussed in the next section.

2.2. Parametric Patches

A mathematical surface can be described in any one of the following three ways:

Explicit:	$z = f(x,y)$
Parametric:	$x = X(u,v),\ y = Y(u,v),\ z = Z(u,v)$
Algebraic:	$F(x,y,z) = 0$

The parametric form is the most popular scheme in graphics applications. Various kinds of parametric surfaces are used. Of these, the bicubic surface is the most widely used in computer graphics.

A bicubic surface patch is defined by third-order (that is cubic) polynomials of two parameters u and v. By varying the two parameters in such a function independently from 0 to 1 we define all the points on a surface patch. If one parameter is assigned a constant value and the other parameter is varied from 0 to 1, the result is a cubic curve.

The form of function $X(u,v)$ used in such a representation is:

$$X(u,v) = UC_xV^T,$$

where $U = [u^3,u^2,u,1]$, $V = [v^3,v^2,v,1]$, and V^T is the transpose of V. The matrix C_x gives the coefficients of the bicubic polynomial. The functions $Y(u,v)$ and $Z(u,v)$ are similarly defined.

Such cubic surface can be expressed in a number of different forms, such as Hermite, Bezier, or B-spline, etc.

Hermite Form

In Hermite form

$$C_x = M_hQ_xM_h^T$$

where

$$M_h = \begin{bmatrix} 2 & -2 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

and

$$Q_x = \begin{bmatrix} x_{00} & x_{01} & \frac{dx}{dv_{00}} & \frac{dx}{dv_{01}} \\ x_{10} & x_{11} & \frac{dx}{dv_{10}} & \frac{dx}{dv_{11}} \\ \frac{dx}{du_{00}} & \frac{dx}{du_{01}} & \frac{d^2x}{dudv_{00}} & \frac{d^2x}{dudv_{01}} \\ \frac{dx}{du_{10}} & \frac{dx}{du_{11}} & \frac{d^2x}{dudv_{10}} & \frac{d^2x}{dudv_{11}} \end{bmatrix}$$

The upper left 2×2 partition contains the four corners of the surface patch, the upper right and lower left partitions specify the slopes along each parametric direction at the corners, while the lower right partition gives the partial derivatives at corners respect to both parameters. These partials are often called the twists. A point on the patch is calculated by blending the geometric data at corners through the bicubic function defined above. The use of a Hermite cubic permits $C^{(1)}$ continuity from one patch to another. Note that the Hermite form of bicubic surface patches is one form of the Coon's patch (*Coon*[25]), so named because it was worked on extensively by the late Steven Coons, an early pioneer in application of computer graphics to Computer-Aided Design. They are also called Ferguson surfaces (*Ferguson*[31]), after another early developer of surface representations.

Bezier Form

The Bezier form of patch (*Bezier*[13][14]) is very similar to the Hermite form, but differs in the definition of the end point tangent vectors. For the Bezier form, 16 points are used. Besides four corner points, 12 additional points are used to specify the tangent vectors of four end points.

The result is:

$$C_x = M_b P_x M_x^T$$

where

$$M_b = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The geometry matrix P which appears consists of 16 control points.

$$P_x = \begin{bmatrix} x_{11} & x_{21} & x_{31} & x_{41} \\ x_{12} & x_{22} & x_{32} & x_{42} \\ x_{13} & x_{23} & x_{33} & x_{43} \\ x_{14} & x_{24} & x_{34} & x_{44} \end{bmatrix}$$

These 16 points are arranged in a 4×4 array. The x_{11}, x_{41}, x_{14}, and x_{44} are x-coordinates of four conner points p_{11}, p_{41}, p_{14}, and p_{44} respectively. The tangent directions at corners are defined by vectors, for example $\overline{p_{11}p_{12}}$ defines the tangent direction along one parameter and $\overline{p_{11}p_{21}}$ the other parameter at corner point p_{11}. Bezier surfaces are attractive for use in interactive design since the control points can be easily manipulated to change the shape of the surface patch.

B-spline Form

The B-spline cubic surface does not in general pass through any control points, but is continuous and also has continuity of tangent vector and of curvature. The term 'spline' traces back to the long flexible strips of metal called splines, used by draftsmen to lay out the surfaces of airplanes and ships.

The B-spline form of surface is analogous to the preceding cases:

$$C_x = M_s P_x M_s^T$$

where

$$M_s = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

Assume that we have a matrix of control points:

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ . & . & \cdots & . \\ p_{m1} & p_{m2} & \cdots & p_{mn} \end{bmatrix}$$

The approximation patch near the points defined by sub-matrix

$$\begin{bmatrix} p_{i,j} & p_{i,j+1} \\ p_{i+1,j} & p_{i+1,j+1} \end{bmatrix}$$

uses

$$P_x = \begin{bmatrix} x_{i-1,j-1} & x_{i,j-1} & x_{i+1,j-1} & x_{i+2,j-1} \\ x_{i-1,j} & x_{i,j} & x_{i+1,j} & xi+2,j \\ x_{i-1,j+1} & x_{i,j+1} & x_{i+1,j+1} & x_{i+2,j+1} \\ x_{i-1,j+2} & x_{i,j+2} & x_{i+1,j+2} & x_{i+2,j+2} \end{bmatrix}$$

With B-spline patches, $C^{(2)}$ continuity is obtained from one patch to another.

Beta-spline

The Beta-spline is a new curve and surface representation (*Barsky*[9]) developed expressly for computer graphics and computer aided geometric design and modeling. Its graphical and geometrical orientation is reflected in the replacement of the traditional parametric continuity, C^2, with geometric continuity, G^2.

From a user standpoint, the Beta-spline can be specified in an intuitive manner through the use of shape parameters β_1 and β_2 called bias and tension, respectively. The shape parameters have an intuitive connection to the "flatness" of the curve and surface, and thus can be used to provide convenient control of shape.

In addition to the shape parameters, a Beta-spline is also controlled by the positions of an arrangement of control vertices. One property of the Beta-spline representation is the capability of modifying one portion of the shape by moving a vertex without altering the remainder of the shape.

Additional Comments

In general fewer points are needed to approximate a curved surface by using parametric patches than by using polygon meshes for a given tolerance. Hence from the point of view of representation, parametric surface is a more efficient method. Therefore, from the point of view of image synthesis, all existing algorithms to render a parametric patch are much more complicated than those to render a polygon mesh (*Blinn*[16], *Catmull*[20], *Lane, et al*[56], and *Whitted*[96]).

Mathematically defined functional surfaces are not only used in surface interpolation, but also used in automatic generation of blending surfaces for intersecting volumes and surfaces (*Hoffmann & Hopcroft*[44], *Middleditch & Sears*[66], *Rockwood & Owen*[83], *Rossignac*[85][86], and *Requicha*[86]). For instance, Hoffmann and Hopcroft presented a general method for automatically deriving a blending surface which smoothly connects two primary surfaces, given that the primary surfaces are quadrics.

2.3. Volumetrics and Octree

The volumetric representation of objects, sometimes called the spatial enumeration representation, is a three-dimensional array. Each element of this array, often called a voxel, corresponds to a cubic volume in space and has a value either one or zero. Like the 2D polygonal patch technique, the 3D volumetric representation can be arbitrarily accurate but is very storage consuming if the resolution of the model is high. This is so much the case that it is still impractical to store a high resolution 3D image on the computers today even given increasingly large memory. Some techniques, such as run-length encoding, have been developed to compress the data of this representation. But they have disadvantages, such as making the transformations difficult.

A related hierarchical volume decomposition method, known as an octree, is another way to represent volume cells (*Gargantini*[36][37], *Meagher*[65], and *Wen & Ahuja*[95]). Each node in the tree corresponds to a region of the universe and has one or more values which define the region. If the value of the node coincides completely with the region, it is a terminal node or a leaf. Otherwise, an ambiguity exists and the node points to eight children that represent the eight subregions or octants of parent node.

This is a very efficient and compact form to represent 3D volumes. It uses only one simple primitive shape, the cube. An arbitrary object can be represented to the precision of the smallest cube. Operations such as hidden surface display and interference detection show only linear growth (*Meagher*[65]) because all objects are kept spatially pre-sorted at all times. Boolean operations benefit from the use of this structure in that the algorithms for them simply traverse the two (or more) input trees in order while generating the output tree. The calculation of object properties such as mass and center of gravity is greatly facilitated by the hierarchical structure.

However, there are also disadvantages in using octrees. When used to represent objects with curved surfaces, the pictures generated have generally poor shadings (*Doctor & Torborg*[28], *Frieder, et al*[34]). This is because the basic elements of this model are six-faced cubes of different sizes. When a geometric transformation is applied to this representation a large amount of fragmentation is usually generated in the transformed representation. The resulting representation is inconsistent in two senses. First, after a rotation operation the new octree generally has a different structure from the original octree. Second, when we apply two geometric transformations, one of which is exactly the inverse of another, in sequence to an octree we generally get a different structure from the original one.

2.4. Volume Density

Volume density can be specified in two ways: sample array and density functions. The sample array scheme is similar to that of volumetric repretation except that each voxel in the sample array is a sample of a continuous density function rather than a binary value. The current techniques for displaying surfaces from volume density data in sample array consists of applying a surface detector to the sample array, fitting geometric primitives to the detected surfaces, then rendering these primitives using conventional surface-rendering algorithms. The techniques differ from one another mainly in the choice of primitives and the scale at which they are defined.

The simplest method to display a volume defined by densities is by applying thresholding to the volume data. The resulting binary representation can be rendered by treating 1-voxels as opaque cubes (*Herman & Liu*[43]). Some early techniques (*Keppel*[53]) connect contours on consecutive slices with triangles. But ambiguities arise when more than one contour of surface exists on a slice. *Lorensen*[61] applies an operator, called the marching cube operator, to locate and create triangles. Substantial improvements in surface shading can be achieved by using local gradient at each vertex which can be calculated locally from density data.

There are other techniques, called volume rendering techniques (see, e.g. *Lovey*[58]), by which it is not necessary to fit geometric primitives to the sampled data. Images are formed by directly shading each sample and project it onto the picture plane. Surface-shading calculations are performed at every voxel with local gradient vectors serving as surface normals. In a separate step, surface classification operators are applied to compute a opacity for every voxel. The rendering colors and opacities are composed from back to front along viewing rays to form an image.

When a volume density is described by density functions, surface points and normal vectors at each surface point are usually given in an analytical form. When surfaces are defined implicitly by algebraic density functions, $F(x,y,z) = 0$, numerical methods are commonly used to evaluate positions of surface points along viewing rays (*Blinn*[15]).

The above volume density techniques are applied mostly to visualizing electron density maps by using isovalue contour surfaces for molecular graphics, and to visualizing computer tomography (CT) data by using region boundary surfaces for medical imaging.

2.5. Particle Systems

The particle system technique introduced by *Reeves*[81] is a method for modeling fuzzy objects such as fire, clouds, and water. A particle system models an object as a cloud of particles that define its volume. Over a period of time, particles are generated into the system, move and change its form within the system, and die, leaving the system, all in a manner controlled by stochastic processes. A particle system can represent motion, changes of form, and dynamics in a manner that is difficult to do with classical surface-based representations. The particles can easily be motion-blurred, and therefore do not exhibit temporal aliasing or strobing.

CHAPTER 3

The Fuzzy Disk Model

3.1. Three-dimensional Oriented Disks

Our search for a new surface representation starts from polygon meshes and bicubic parametric patches. But we need a form of three-dimensional surface patch which is simpler than these, one that uses less storage and makes computations on it in the image synthesis process simpler and more efficient.

We offer the following model to meet these conditions. A three-dimensional oriented disk is simpler than either a polygon or a bicubic parametric patch. Collections such disks turn out to define a very suitable representation for graphic modeling. Specifics will now be developed. A three-dimensional oriented disk can be described by its center coordinates (x,y,z), the radius of the disk, and a normal vector (n_x,n_y,n_z) defining the orientation of a disk. Any set of such disks can be written as

$$S = \{p_1,p_2, \cdots ,p_n\},$$

where each p is defined as

```
struct p {
      float   x,y,z;
      float   n_x,n_y,n_z;
      float   radius;
      float   r,g,b;
}
```

Here x,y,z are coordinates in Cartesian system;

n_x,n_y,n_z are the components of surface normal vector at the sample position;

radius is the radius of the disk;

r,g,b are red, green, and blue color intensity values.

As a first indication of the ability of such collections to represent arbitrary surfaces graphically, we offer Figure 3.1, which illustrates a hemisphere generated from a collection of three-dimensional oriented disks covering the hemisphere.

To explore the idea just introduced, we must explore the following questions:

(1) How can we generate smoothly curved surfaces from collections of oriented circular patches?

(2) Are there efficient ways of organizing such models from both the point of view of data representation and the point of view of image regeneration?

(3) Does this model have substantial advantages over other existing models?

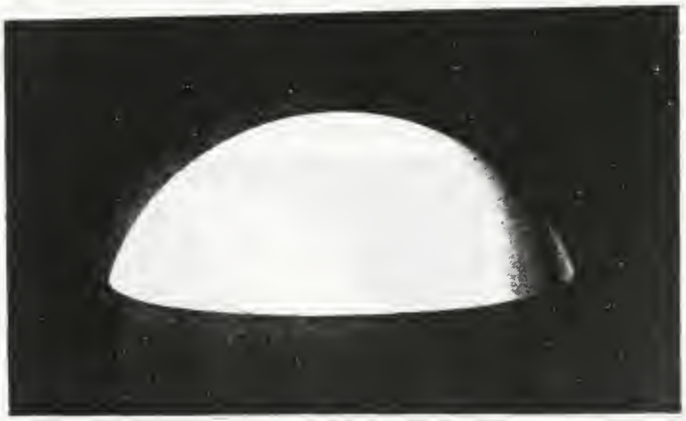


Figure 3.1

(4) What are disadvantages of this model?

(5) How do we construct circular patch models from empirically sampled three-dimensional surface data?

The remainder of this chapter initiates discussion of these questions.

3.2. From Planar Surfaces to Curved Surface - Local Disk Blending

Since a disk is a planar surface, a smooth curved surface approximated by a number of disks will appear in a synthesized image as many planar facets if nothing special is done; this is a known disadvantage of polygonal models. This problem is illustrated in Figure 3.2 which gives a close view of a small collection of disks.

However, something better from the graphical point of view can be done, using an idea analogous to that of spline. In splines, control points are provided and parametrically defined "blending" functions interpolate points between those control points by "blending" geometrical data taken from the control points to form a continuously smooth curved surface.

A similar weighted blending method based on overlapping disks can be introduced. This

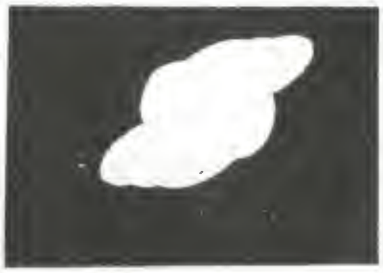


Figure 3.2

method while is analogous to, also differs significantly from traditional splines in both its mathematical definition and its intuitive geometrical explanation, which is as follows:

> *Interpolation in our model is not based on "blending" the geometric data at discrete control points as in traditional spline surfaces, but is based on blending density fields defined with disks in space.*
>
> *A surface is not defined by explicit parametric functions as in the traditional spline surfaces, but is defined in more implicit fashion in our method.*
>
> *The traditional spline surfaces generally require points arranged in specific sequential order, whereas we don't require any topological connections between blending disks at all.*

However our model shares a common goal with the traditional spline models which is to interpolate points between a given set of points to form a smoothly curved surface.

To develop these thoughts we start by presenting a view-independent local disk blending method which emphasizes the mathematical simplicity and generality of the ideal method. After this we develop an approximation method, useful for geometric rendering, which is more efficient than the first method, but which can occasionally be inconsistent in that slight irregularities can occur when viewing position is changed.

3.2.1. Local Disk Blending: View Independent Definition

Instead of treating a disk as an explicit planar surface patch, we will define a density field in three-dimensional space about a disk in a functional form, and from this define surfaces implicitly by a density equation.

Assume we have a disk of radius R in three-dimensional space, centered at point O with normal vector $\overline{n}$, as shown in Figure 3.3. A fuzzy region is defined by this disk in the following manner. Given a point P in space, let

$$\overline{V_1} = \overline{P} - \overline{O} \tag{1}$$

which is a vector from O to P. We then separate $\overline{V_1}$ into two components $\overline{V_2}$ and $\overline{V_3}$ such that $\overline{V_2}$ is the projection of $\overline{V_1}$ onto $\overline{n}$, i.e.

$$\overline{V_2} = (\overline{V_1} \cdot \overline{n})\overline{n} \tag{2}$$

and

$$\overline{V_3} = \overline{V_1} - \overline{V_2} \tag{3}$$

which is a vector orthogonal to $\overline{V_2}$.

Let

$$r = \frac{|\overline{V_3}|}{R} \tag{4}$$

be the normalized distance from a point on the disk to its center, and let

$$v_2 = \overline{V_1} \cdot \overline{n} \tag{5}$$

Then the corresponding density at the point P is defined by function

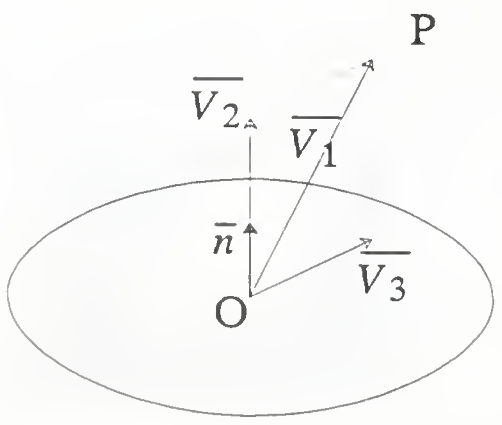


Figure 3.3

$$f_d(\bar{P}) = v_2 w(r) \tag{6}$$

where $w(r)$ is some weighting function, to be chosen as described below, which has a range from 0.0 to 1.0.

What should the weighting function be like? In general, we need larger weights at the points near the center of the disk and smaller weights at the points far from the center of the disk, so a disk can have a large influence on the portion of a blending surface which is near the disk's center, the control becoming weaker as the r defined above increases, and finally the control diminishes when the r is large enough.

In the density field defined by (6) we also wish all points on the side of a disk to which the normal vector of the disk is pointing to have positive density values, while on the other side of the disk, all points have negative density values. On the plane passing through the disk, density values will then be zero. Also, along the direction of normal vector, density will increase monotonically.

We can combine densities for a collection of fuzzy disks by summing the contributions of separate disks:

$$f(\bar{P}) = \sum_i v_{2i} w(r_i) \tag{7}$$

This leads to the following overall **Surface Definition:** *Let* $\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z})$ *be the gradient of the density function (7). Then the resulting blending surface is defined as the collection of all those points where the following conditions are satisfied:*

(1) $f(\bar{P}) = 0$*;*

(2) $|\nabla f| > \epsilon$*, where* $\epsilon > 0$ *is an auxiliary small positive number;*

(3) $f(\bar{P} + \nabla f \delta) > 0$ *and* $f(\bar{P} - \nabla f \delta) < 0$*, where* $\delta > 0$*, is also a small positive number;*

(4) For some i, $r_i \leq 1$*.*

The first condition above states the essential equation defining our notion of blending surface. The second condition ensures that local gradient in a density field should be sufficient large (The necessity of this condition will be seen once the weighting function that we use is defined). The third condition guarantees the existence of a local "zero-crossing" from one side of a surface to the other side through any surface points. The final condition restricts a blending surface to be defined only near the disks which define it.

Since a surface is defined implicitly by an equation (plus three other conditions), our technique for finding a surface point along a viewing ray will be numerical. We will discuss the problems that this causes in Chapter 4.

First, however, we consider two candidates for the weighting function $w(r)$:

(1) Gaussian distribution function

(2) Cubic weighting function

3.2.2. Weighting Function I: Gaussian Distribution

The Gaussian distribution function is

$$g(x) = e^{-a_0 x^2} \tag{8}$$

where a_0 is standard deviation factor. Figure 3.4 shows the well-known form of this function (where $a_0 = 1$).

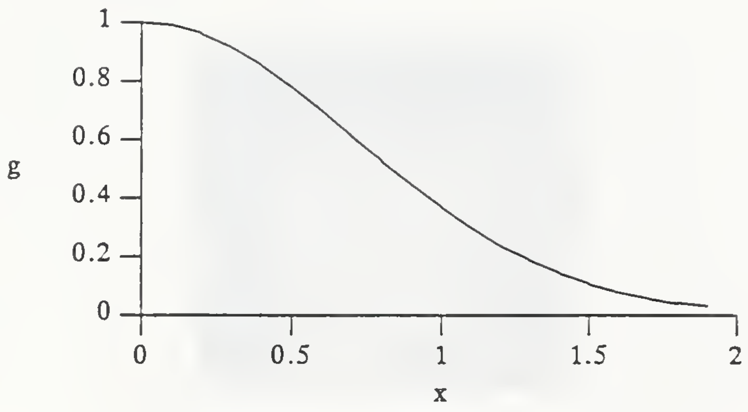


Figure 3.4

Figure 3.5 shows a cross section of the density field defined by a disk if we replace the $w(r)$ in (6) by (8), along an intersection plane placed perpendicular to the disk and passing through the center of the disk. In this figure, the gray value at each pixel in the picture is proportional to the magnitude of density value; red represents positive and blue represents negative.

Figure 3.6 shows another cross section of such field. This picture shows density values

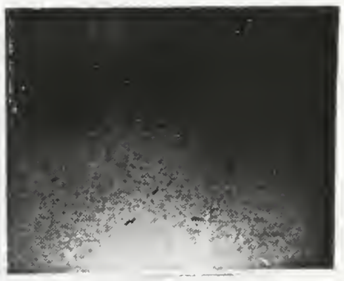


Figure 3.5

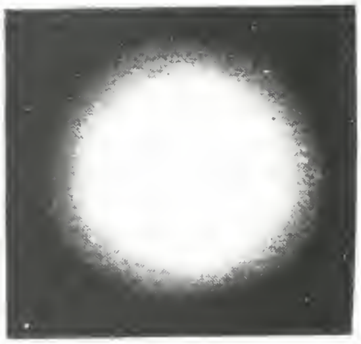
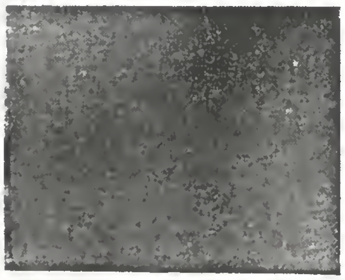


Figure 3.6

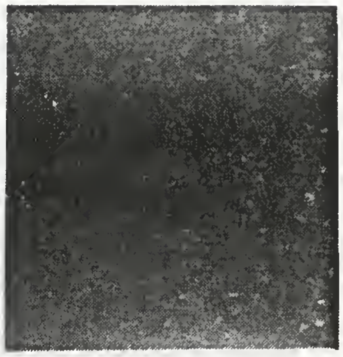
along an intersection plane which is parallel to the disk. Here the density value is defined by a typical Gaussian distribution.

Figure 3.7 (a), (b), and (c) show the density maps of three disks respectively placed in different positions in space. (Note that all three disks are perpendicular to the plane of density maps). Then Figure 3.7 (d) shows the blending density field of these three disks, i.e. the summation of three previous density maps.

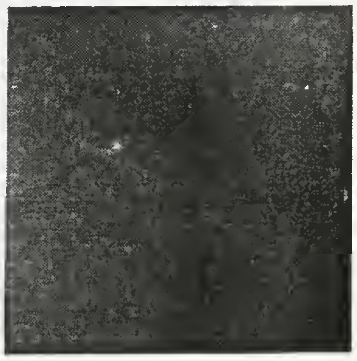
In Figure 3.7 (d) we have a set of points along a smooth curve which satisfy the surface definitions given in Section 3.2.1. Figure 3.8 shows several surfaces generated by blending



(a)



(b)



(c)

(d)
Figure 3.7

several disks.

The Gaussian distribution function $g(x)$ approaches 0 when $x \to \infty$. But it is always posi-

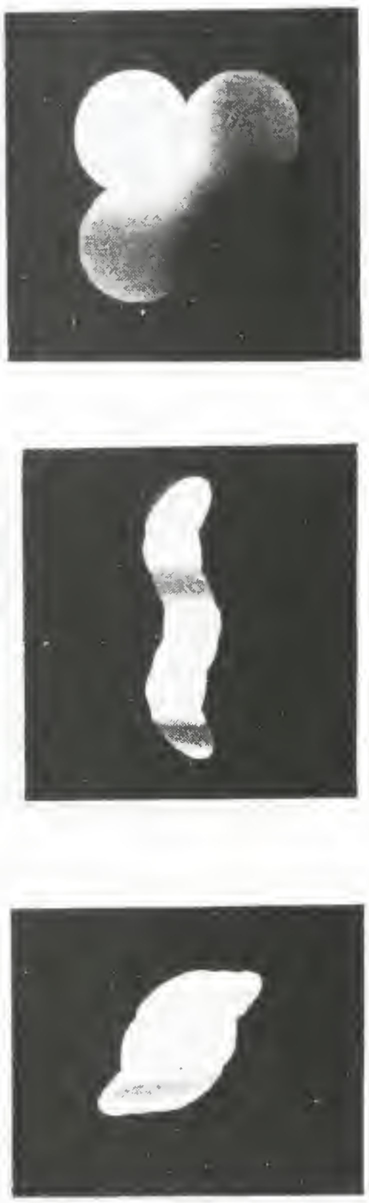


Figure 3.8

tive. Thus if the Gaussian distribution is used as the weighting function, then for $r > 1.0$ the weights will be very small, but be still greater than zero. This means that the effect of a disk on the shape of a blending surface extends indefinitely far beyond the radius of the disk. This property is undesirable for two reasons:

(1) When we evaluate a surface point in a density field, we must count the contributions from all given disks in space. This causes the algorithms which handle this model to be very time-consuming.

(2) For the purpose of efficiency in practical applications, if we count only the strong contributions from near disks, but discount the weak contributions from disks far away, a potential problem arises: possible discontinuities in the generated blending surfaces.

For these reasons, in the next section we study another possible weighting function which eliminates the problem with Gaussian distribution described above.

3.2.3. Weighting Function II: Cubic

A cubic weighting function is defined as:

$$c(x) = \begin{cases} 1-3x^2+2x^3 & 0 \le x < 1.0 \\ 0 & 1.0 \le x \end{cases} \tag{9}$$

A curve defined by this function is shown in Figure 3.9.

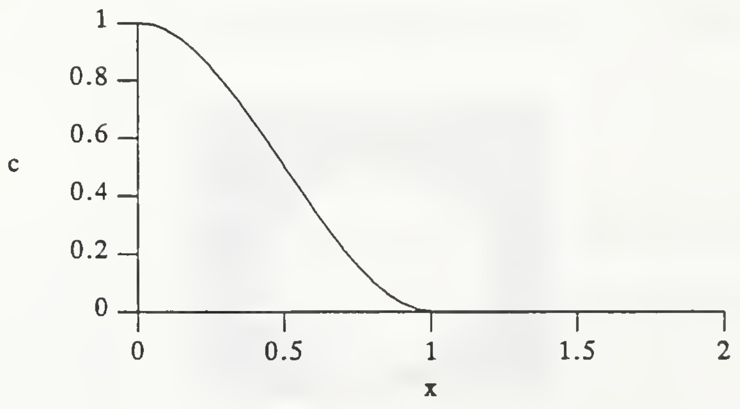


Figure 3.9

For this cubic weighting function, $c(x) = 0$ for $x > 1.0$. Notice also that the first derivative of this function is continuous for $x > 0$. This ensures that the control of a disk over the shape of a blending surface will be rapidly reduced as the r approaches 1.0 and completely disappears after $r > 1.0$. Replacing the $w(r)$ in (6) by (9) gives a similar but slightly different density distribution from the previous Gaussian distribution. Figure 3.10 shows a cross section of the density map on an intersection plane perpendicular to a disk and passing through the center of the disk, and Figure 3.11 shows another cross section which is parallel to the disk.

Figure 3.12 illustrates some blending surfaces generated by using cubic weighting function.

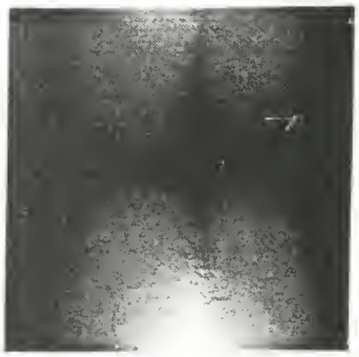


Figure 3.10

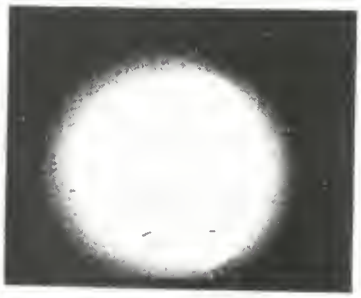
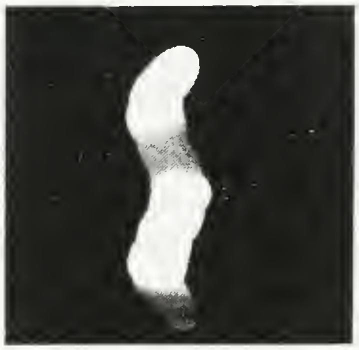
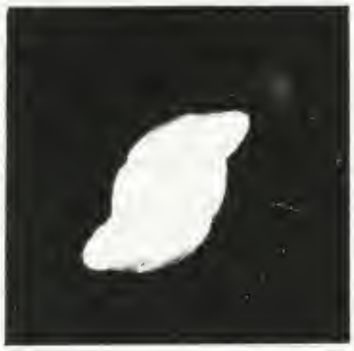


Figure 3.11



(a)



(b)
Figure 3.12

3.2.4. Locality

The control of a disk over the shape of a blending surface is localized: recall the blending density density function (7)

$$f(\bar{P}) = \sum_i v_{2i} w(r_i)$$

The contribution of disk i to $f(\bar{P})$ is non-zero for $r_i < 1.0$, and is zero for $r_i \geq 1.0$ in case of cubic weighting function been used, or is very small for $r_i \geq 1.0$ in case of Gaussian distribution function been used.

Obviously then, from the geometric point of view, a disk has control over the shape for only a portion of a blending surface which intersects with the cylindrical volume of the disk.

3.2.5. Continuity

In this section we examine the continuity of a blending surface, and the continuity of first derivative of a blending surface, with two different weighting functions.

A blending surface in our model is continuous. This can be shown by the continuity of blending density field

$$f(\overline{P}) = \sum_i v_{2i} w(r_i).$$

To show $f(\overline{P})$ is continuous, it is sufficient to show that v_2 is continuous and $w(r)$ is continuous. Recall (5)

$$v_2 = \overline{V_1} \cdot \overline{n}$$

$$= (\overline{P} - \overline{O}) \cdot \overline{n} \qquad (10)$$

Let

$$\overline{P} = (x, y, z), \qquad (11)$$

$$\overline{O} = (x_0, y_0, z_0), \qquad (12)$$

and

$$\overline{n} = (n_x, n_y, n_z). \qquad (13)$$

By substituting (11), (12), and (12) into (10), we have

$$v_2(x,y,z) = (x - x_0)n_x + (y - y_0)n_y + (z - z_0)n_z \qquad (14)$$

and it is continuous.

By a similar calculation, we have

$$r = \frac{1}{R}\sqrt{(v_1^2 - v_2^2)} \qquad (15)$$

where

$$v_1^2(x,y,z) = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \qquad (16)$$

and we always have

$v_1^2 \geq v_2^2$.

So r is continuous.

On the other hand, both weighting functions, $g(r)$ and $c(r)$ are continuous. Hence, the composition of any of these weighting function with function r is continuous.

Following some similar elementary calculations, we have also continuous derivatives of these functions.

3.2.6. Shape Control: Smoothness and Flatness

By the definition of blending surfaces given in 4.2.1, the shape of a blending surface for a given set of three-dimensional disks is uniquely defined if all constants in the blending density functions are fixed. Remember that, a disk in this model is not treated simply as a surface patch, but instead defines a fuzzy density field which is a three-dimensional oriented cylindrical shaped region. The density function for a three-dimensional disk is

$$f_d = v_2 w(r)$$

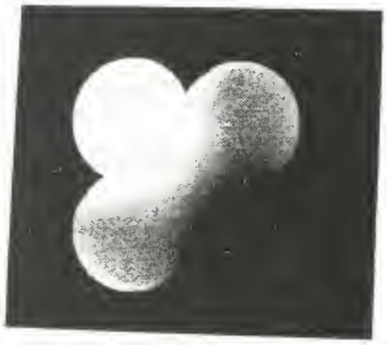
which is composed of two parts: magnitude, v_2, and weight, $w(r)$. It is essential to know what are important factors which affect the shape of a blending surface. Specifically, we want to study the relationships between the constant parameter of the density function and the shape of the generated blending surface, and how the shape of a blending surface can be controlled by varying the constant parameter of the density function. First we study this problem using Gaussian distribution as the weighting function. We will show how the results from this model can be extended to other two weighting functions.

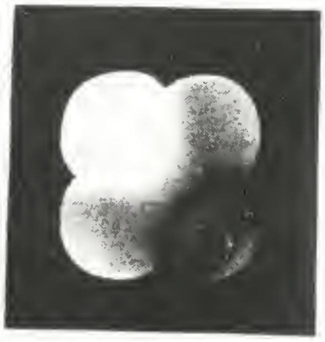
The density function using Gaussian distribution as its weighting function for a single disk is

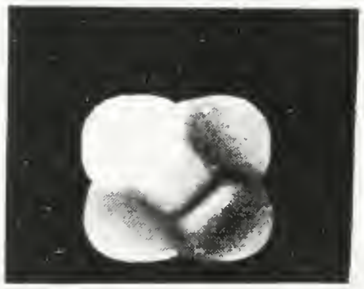
$$f_d = v_2 e^{-a_0 r^2}$$

By varying the constant, a_0, we wish to see whether the shape of a blending surface will be changed and, if so how it changes.

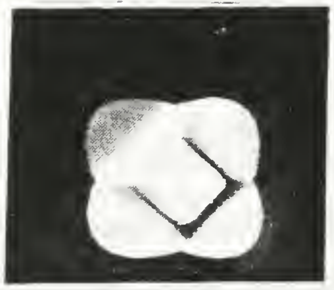
Figure 3.13 (a), (b), (c), and (d) are generated with $a_0 =$ 0.5, 1.0, 2.0, and 4.0, respectively. By decreasing a_0 - the standard deviation of Gaussian distribution, we see from these pictures that the blending surface becomes more smoothly curved. On the other hand, if we increase a_0, the blending surface becomes flatter in each sub-region of the surface, and eventually approaches a polyhedral surface (while intersection edges are still locally smoothly curved).


(a)


(b)



(c)



(d)

Figure 3.13

This phenomenon can be explained as follows. Let us define an axis of a disk to be the line parallel to the normal vector of the disk and passing through the center of the disk. Assume we have two disks in space; let l_1 be the axis of disk 1, and l_2 be the axis of disk 2. For a given point P in the space, let R_1 be the distance from P to l_1, and R_2 be the distance from P to l_2, respectively. We can draw density curves with two disks with respect to point P as shown in Figure 3.15. The density value at the point P is the sum of the contributions from the two disks.

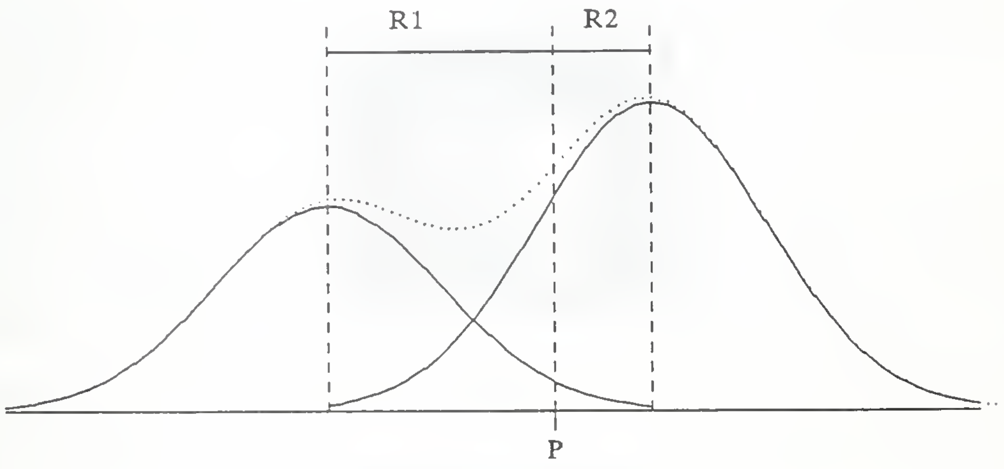


Figure 3.14

By decreasing the standard deviation a_0, Gaussian distribution curves are stretched horizontally which results in an increasing overlapping of two curves as shown in Figure 3.15. This means that the weights of both disks at such a point are increased but the changes are not linearly proportional. In this case, the two weights are comparable. So the contributions of two disks are both important in such a situation, and the density at P is a blend of two disks.

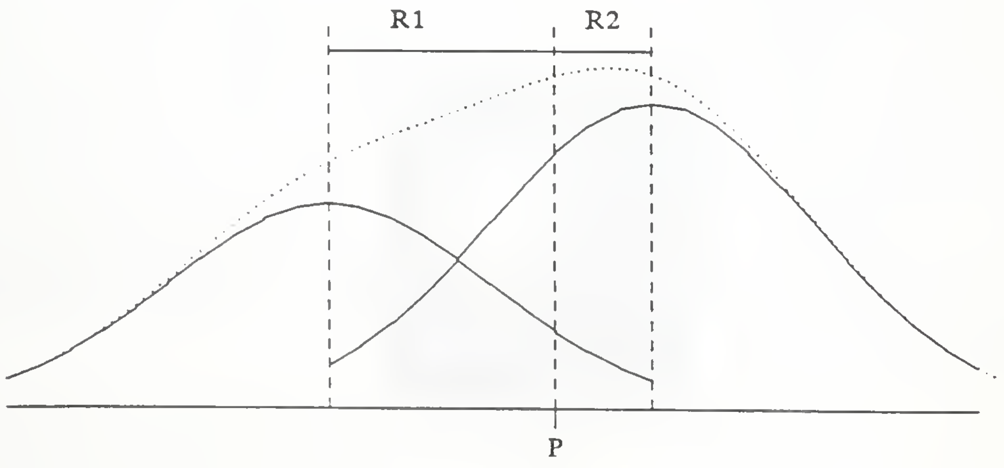


Figure 3.15

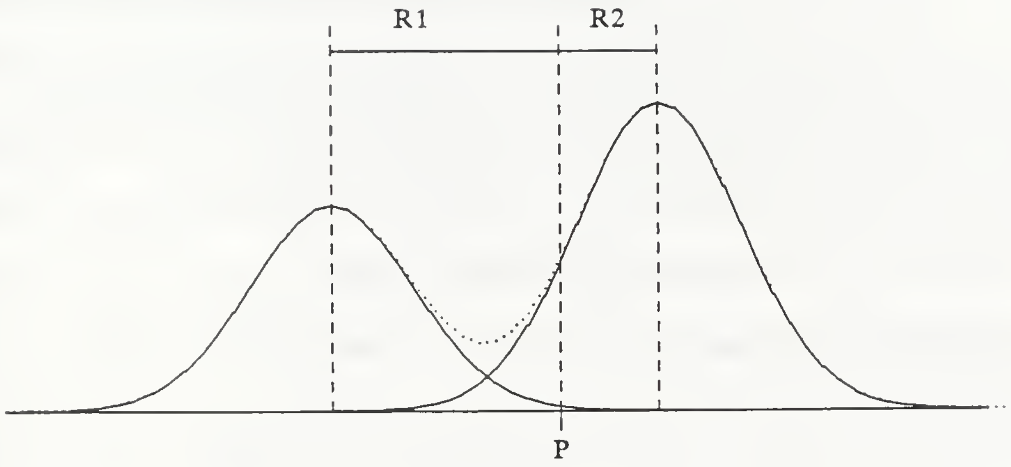


Figure 3.16

On the other hand, if a_0 is increased, the Gaussian distribution curves are compressed horizontally, and the overlapping of two curves are decreased. Particularly in a certain range of a_0, the contribution from disk 1 is much stronger than that of disk 2 as shown in Figure 3.16. This implies that the density distribution at the point P is primarily determined by disk 1. For a_0 larger than a certain value, two curves become effectively separated; there will be little or no blending effects. In such a situation, a portion of blending surface near the center of a disk is primarily determined by this disk while other disks have very weak control over this portion.

We observe that varying a_0 is equivalent to changing the sizes of disks. This is because $a_0 r^2 = a_0 \frac{v_1^2 - v_2^2}{R^2}$, so the effect of any change in a_0 can be achieved equivalently by adjusting R with remaining a_0 unchanged. Increasing a_0 is equivalent to decreasing the radius R of the disks, and vise versa. The larger the radius of disks, the larger is the overlapping between disks, and the stronger the blending effects will be.

The cubic weighting function does not contain a constant parameter like the a_0 of the

Gaussian distribution. But if we modify the definitions of original weighting functions slightly, we can achieve the similar effects. Notice that the effects of increasing or decreasing a_0 in Gaussian distribution are actually the compression or stretching of a characterizing curve in the horizontal direction. For the cubic weighting function,

$$c(x) = \begin{cases} 1-3x^2+2x^3 & 0 \le x < 1 \\ 0 & 1 \le x \end{cases}$$

We can replace x by $a_0 s$ where a_0 is a constant and s is a new variable, we have

$$c(a_0 s) = \begin{cases} i1-3(a_0 s)^2+2(a_0 s)^3 & 0 \le s < \frac{1}{a_0} \\ 0 & \frac{1}{a_0} \le s \end{cases} \tag{17}$$

Equivalently, we can define a new function:

$$c_a(x) = \begin{cases} 1-3a_0^2x^2+2a_0^3x^3 & 0 \le x < \frac{1}{a_0} \\ 0 & \frac{1}{a_0} \le x \end{cases} \tag{18}$$

By increasing a_0, a weighting curve is compressed, and vise versa; see figure 3.17.

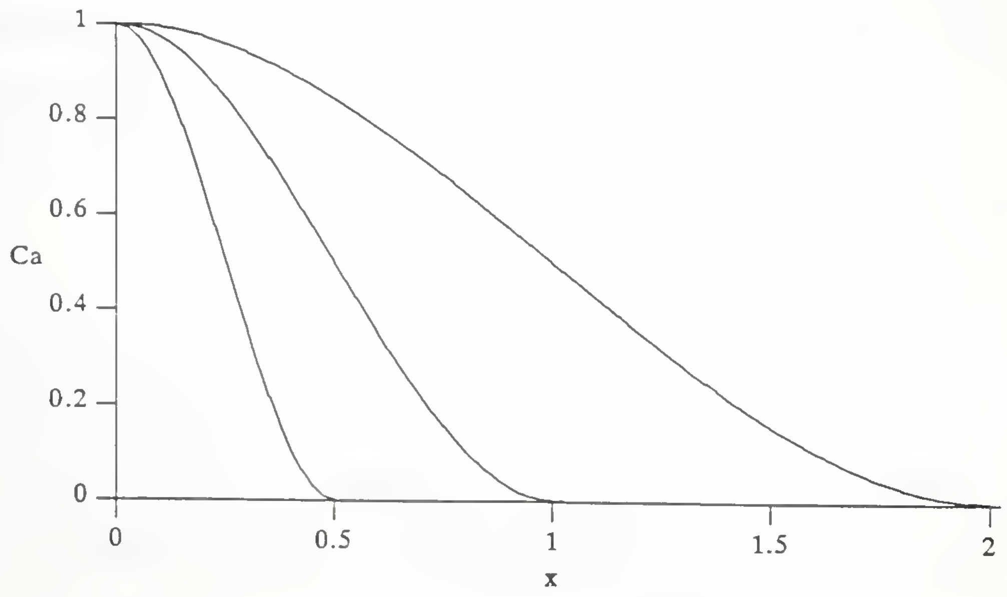


Figure 3.17

3.2.7. Local Disk Blending: A View Dependent Method (Approximation)

The blending method described in the last section is rotationally invariant, that is, independent of view point. The surface defined by a given set of disks is consistent no matter from which direction you look at it. In this section we describe an approximation method to the previous one. This approximation method is said to be view dependent because errors introduced by this method vary as the view point changes. Though this approximation method generates a certain amount of error in surface generation, it allows a much more efficient calculation than the previous method.

Assume that we have a disk d of radius R centered at point $O=(x_o,y_o,z_o)$ in space, and that a viewing ray L intersects the disk at point $p=(x_p,y_p,z_p)$. Let

$$S = \frac{s}{R} = \frac{1}{R}\sqrt{(x_p-x_o)^2+(y_p-y_o)^2+(z_p-z_o)^2} \tag{19}$$

We use the previously defined weighting function $w(r)$, or equivalently $w(p)$, here again.

If a viewing ray L intersects with one or more disks, we define the z value of the intersec-

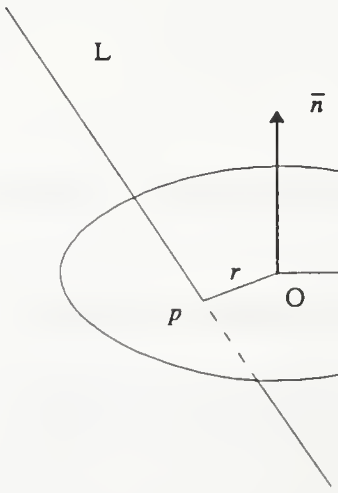


Figure 3.18

tion point of the blending surface and the viewing ray to be

$$z = \frac{\sum_i z_{p_i} w(p_i)}{\sum_i w(p_i)} \tag{21}$$

That is, we sum up weighted z_p from each intersecting disk, then divide this weighted sum by the total weight (a normalization step) to get the blended z.

The difference between this method and previous method is the following. In the view independent method, each disk affects the blending surface by pulling the blending surface towards itself along the direction parallel to its normal vector, while in view dependent method, each disk affects the blending surface by pulling the blending surface towards itself along the viewing ray. Obviously, as a viewing ray changes its direction, the blending surface changes its shape. This is why the latter method is called view dependent.

In the view independent method, since a surface is defined implicitly, numerical methods are generally used to find a surface point along a viewing ray (it will be discussed in the next chapter). But in this view dependent method, we do not need iterations which are usually performed in numerical methods. We now only have to count all disks which intersect with a viewing ray and compute a weighted sum directly.

Figure 3.19 shows a blending surface generated by the new approximation method.

3.3. Multi-resolution Structure and Adaptive Sampling

In Section 3.1, the structure of a collection of three-dimensional oriented disks is simply a non-ordered linear sequence. But is there any more efficient structure for this three-dimensional disk data representation? In this section we introduce a multi-resolution structure for the data representation. This structure is analogous to octree hierarchical structure, but

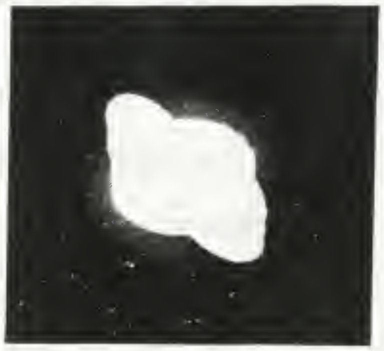


Figure 3.19

with some differences.

We review the octree notion. In each level of an octree, the volume elements are identically sized cubes. The volume of a cube at any level (except the root) is one eighth of that in its immediate parent level. The greatest advantage of octrees is their ability to refine volume details in an adaptive manner, that is, finer levels are created only when finer approximation is necessary. This kind of hierarchical structure is unnatural for polygons or parametric surfaces mainly because the polygons and parametric surface patches do not carry explicit sizes. However, the three-dimensional disks in our model do have explicit sizes. This suggests that a hierarchical structure similar to octree might be possible for our model.

In real-world scenes, the complexities of both geometrical shape and texture are in general variable from one region to another over the surfaces of the scene. Here, by complexity of geometrical shape we mean essentially the local curvature of a surface, and by complexity of texture we mean the highest frequency of visually relevant texture information. It is desirable to sample curved surfaces in different densities for different regions according to these complexities. In our representation, this can be done by adjusting the size of disks used depending on the local surface and texture complexities. The more complex the surface is in a region, the denser the sampling should be in such a region, and vise versa. Techniques which do this kind of non-uniform sampling are generally known as adaptive sampling methods.

Adaptive sampling is useful since it provides an efficient way to compress a representation. Extra data are created only at places where it is necessary. Besides this, the image generation process in computer graphics will also benefit from this method. In image synthesis, we can render surface elements in an order from coarse to fine and avoid unnecessary detail which will be washed out in the generated image.

Figure 3.20 illustrates that some surface details, such as small bumps of different sizes on a smooth surface, can be refined by smaller disks.

In our model, a collection of three-dimensional oriented disks is organized into a multi-resolution structure according to radius of disks in the following manner. Let $R_{\max}$ be the largest radius of disks in a given collection. The first level contains all disks whose radius are in the range $[R_1, R_{\max}]$ where $R_1 = \frac{R_{\max}}{2}$. The second level contains all disks whose radius is in the range $[R_2, R_1]$ where $R_2 = \frac{R_1}{2}$, and so on until no more disks are left. Within each level, disks are still stored in a non-ordered manner.

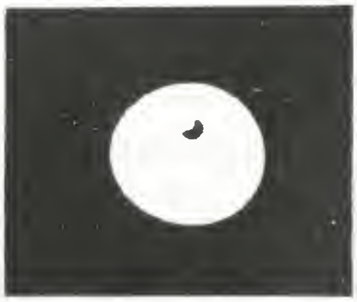


Figure 3.20

Furthermore, if we choose the sizes of disks in a manner similar to the octree method, that is, all disks at a same level have same size, and size of disks at a level is the half of that at its immediate parent level, then we only need the size for disks at the top level while the radius with all individual disks can be omitted. This further reduces the storage requirement of the representation.

Subdivision by Orientation

For the purpose of rendering efficiency, we can go further by dividing our data into subgroups according to the orientation of sample disks within each level of the multi-resolution representation just outlined. The disks in those groups which are totally facing away from the viewer are not visible from the view point. Thus, they can be ignored during the rendering process. This allows us to cut off some of these subgroups before rendering them, for the purpose of efficient hidden surface removal.

3.4. Algorithm to Construct the Surface Representation from Sampled Data

We now describe an algorithm for constructing a disk-based representation of the kind that we have described from sample data.

Two auxiliary maps are computed first. One is a curvature map in which each value represents a kind of local curvature of corresponding sample points. The second map is texture map. These two maps are used in our algorithm for determining the size of disks in different regions.

Figure 3.21 is a radius map of a Haydn bust acquired by a rotational laser scanner (see Appendix A).

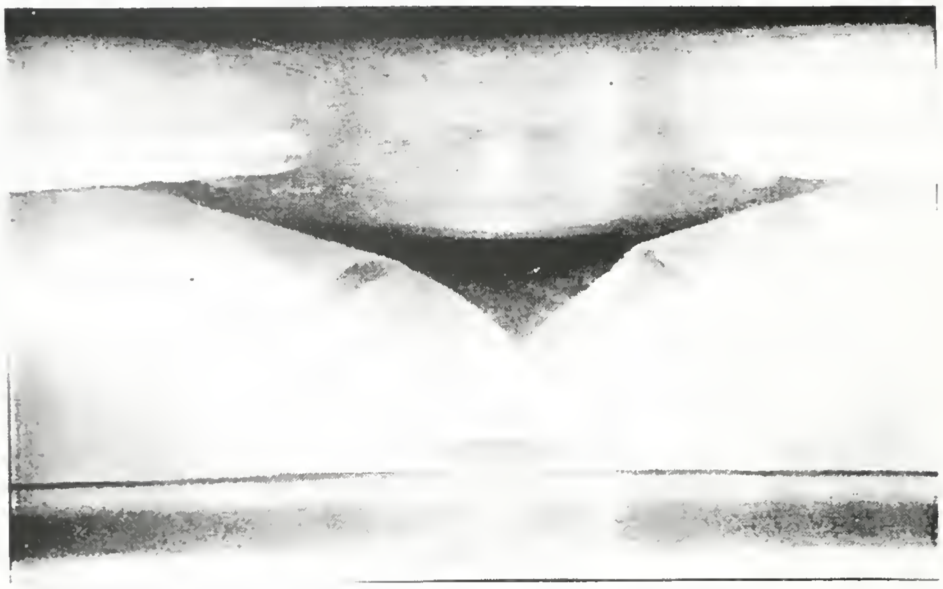


Figure 3.21

Figure 3.22 shows the curvature map corresponding to the radius map of the Haydn bust. In this map, curvature values are quantized into four levels. Figure 3.23.a shows a region

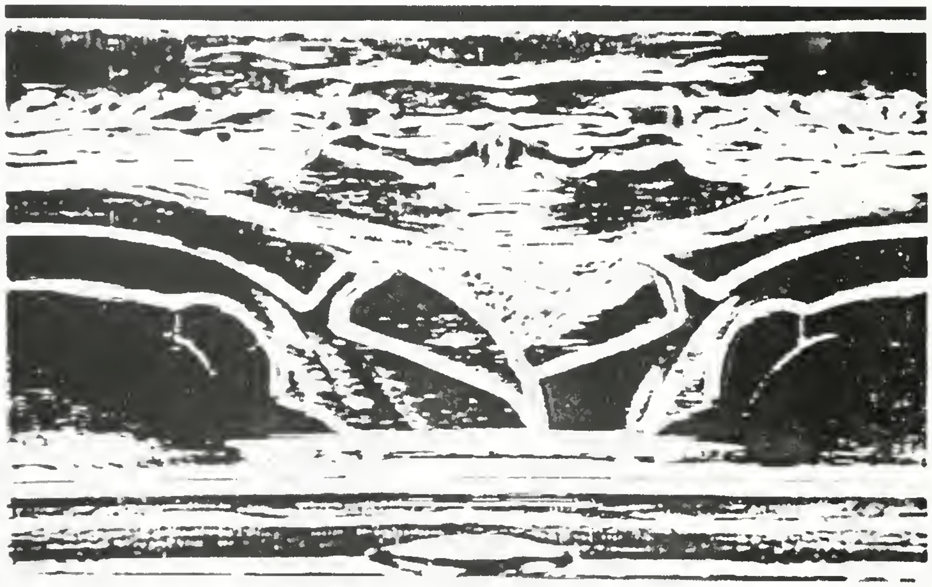
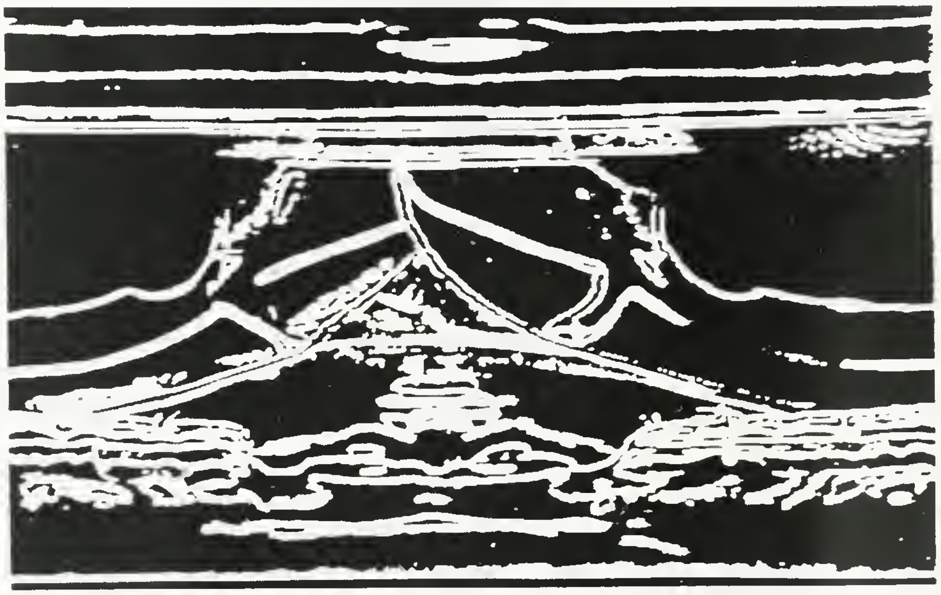


Figure 3.22

mask for the fourth level at which the curvature is the highest. Figure 3.23.b shows a region mask for the third level and the resolution of this map is half of the fourth level. Similarly, we have region masks for the second and the first levels as shown in Figure 3.23.c and Figure 3.23.d respectively. So we get a pyramid of region masks. The sampling density at a lower level is twice as dense as that in its next higher level, and the radius of disks used for a lower level is half of that for its next higher level.



(a)

(b)

(c)

(d)

Figure 3.23

A texture map can be similarly constructed, and be applied to the representation construction process.

The orientation of a disk is computed locally. First we compute two orthogonal tangent vectors at a given point $p_{i,j}$. Along the direction of index I, we take the normalized vector

$$\overline{\delta I_1} = \frac{p_{i,j} - p_{i,j-1}}{|p_{i,j} - p_{i,j-1}|}$$

and

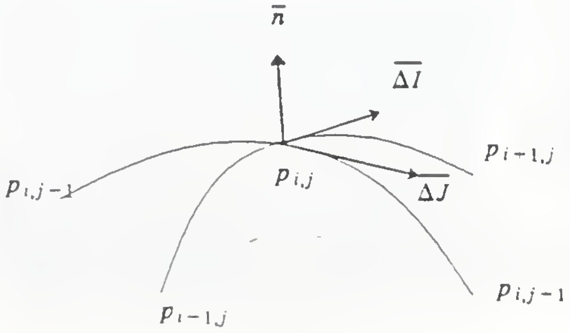


Figure 3.24

$$\overline{\delta I_2} = \frac{p_{i,j+1} - p_{i,j}}{|p_{i,j+1} - p_{i,j}|}$$

to be two tangent vectors along index I on both sides of $p_{i,j}$. Now we define the tangent vector at $p_{i,j}$ along index I to be the average of above two vectors:

$$\overline{\Delta I} = \frac{1}{2}(\overline{\delta I_1} + \overline{\delta I_2}) \tag{22}$$

Similarly, we have a tangent vector along index J

$$\overline{\Delta J} = \frac{1}{2}(\overline{\delta J_1} + \overline{\delta J_2}) \tag{23}$$

where

$$\overline{\delta J_1} = \frac{p_{i,j} - p_{i-1,j}}{|p_{i,j} - p_{i-1,j}|}$$

$$\overline{\delta J_2} = \frac{p_{i+1,j} - p_{i,j}}{|p_{i+1,j} - p_{i,j}|}$$

Then the normal vector at $p_{i,j}$ is defined as

$$\bar{n} = \overline{\Delta I} \times \overline{\Delta J} \tag{24}$$

The generated disks are then sorted by their size of radius. So we can divide the set into multiple levels with a different resolution at each level. Within each level, disks are further subgrouped by orientation.

3.5. Advantages of the Model

- *Simplicity*

  It is clear that a three-dimensional oriented disk is a simpler surface patch primitive than other existing ones, such as polygons and parametric patches. This makes the handling algorithms of it simpler and faster. For example, ray-surface intersection calculations, the main computation of ray-tracing techniques, are very simple with three-dimensional disks.

- *Generality*

  There is no topological connections between disks. This makes the model to be very flexible for representing complex surfaces.

- *Quality*

  Our fuzzy disk blending method allows us to generate smoothly curved surfaces from overlapping fuzzy disks. The shapes of blending surfaces can be controlled in a geometrically intuitive way.

- *Efficiency*

  The efficiency of three-dimensional disk representation can be viewed from different points.

  First, disks fit naturally into a multi-resolution structure analogous to the hierarchical structure of octrees. This multi-resolution structure provides us an efficient data representation by which we can refine surface details adaptively. From the point of view of dynamic display, such multi-resolution structure will support a coarse-to-fine display in which lower level surface details can be avoided if they are nearly non-recognizable in generated images.

  Second, for the same number of surface elements, the total storage required by three-dimensional disks is usually less than required by polygons or parametric patches, with the exception of the situation where the polygons can be represented by rectangular meshes (this will largely restrict the capability of polygons to represent complex surfaces), the average number of floating-point numbers required per polygon is not less (usually larger) than that per disk. On the other hand, it is of course not fair to compare a single three-dimensional disk with a bicubic parametric patch, because a bicubic parametric patch represents more complicated shape than a single disk. But if we use four disks and place them at the four corners of a Hermit patch or a Bezier patch respectively, then we can get similarly shaped smooth surfaces from both methods. A bicubic patch requires 16×3 floating-point numbers while four three-dimensional disks requires only 4×7 floating-point numbers. So, the storage requirement by three-dimensional disks is usually less than polygons and parametric patches. Or in the other words, given a fixed amount of storage, we can store more disks than other primitive patches and hence we can store finer surface information.

Third, to approximate a smoothly curved surface, the number of disks required is much less than that by polygons.

The fuzzy disk model, however, has also some disadvantages as described below.

3.6. Disadvantages of the Model

• The model is specific for graphics display of complex arbitrary surfaces of real-world objects. No other information, such as mass or volume, about the objects being modeled can be derived from the model, in contrast to solid modeling techniques.

• It is not possible to traverse a surface in our model, in contrast to "winged-edge" polygon representation and octrees.

• Our model is relatively expensive in approximating regular geometric shapes, such as rectangles, in contrast to polygons. Straight edges of polygons need to be refined by many small disks in order to satisfy some given tolerance.

CHAPTER 4

Rendering Algorithms

4.1. Basic Rendering Algorithm

4.1.1. Coordinate System

We define a right-hand coordinate system as shown in Figure 4.1. The view point is placed at the origin. An image plane is perpendicular to the z-axis and is placed at $z = focal$ (note that focal is a negative value.), where *focal* is the focal length of camera.

A viewing ray through a particular pixel in the image plane is defined by a vector

$$\bar{L} = (x_l, y_l, focal) \tag{1}$$

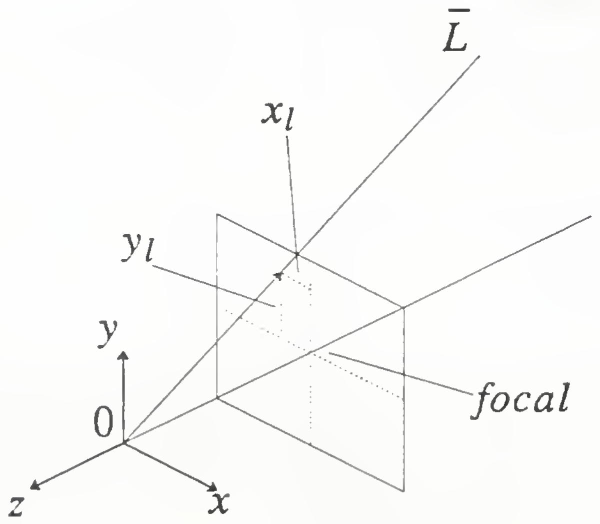


Figure 4.1

Any point on this ray is scalar multiple of this vector and can be parameterized by its z coordinate, yielding

$$(x,y,z) = \frac{z}{focal}\bar{L}$$

$$= (\frac{x_l}{focal}z, \frac{y_l}{focal}z, z) \qquad (2)$$

4.1.2. Parameterization of Density Function by Depth z

Recall the density function introduced in (7) of Chapter 4.

$$f(\bar{P}) = \sum_i v_{2i} w(r_i)$$

where

$$v_2(x,y,z) = \overline{V_1} \cdot \bar{n}$$

$$= (x-x_0)n_x + (y-y_0)n_y + (z-z_0)n_z$$

$$r = \frac{1}{R}\sqrt{v_1^2 - v_2^2}$$

$$v_1(x,y,z) = \sqrt{(x-x_0)^2+(y-y_0)^2+(z-z_0)^2}$$

and

$P = (x,y,z)$ is an arbitrary point in space

$O = (x_0,y_0,z_0)$ is the center of a disk

$\bar{n} = (n_x,n_y,n_z)$ is the normal vector of a disk.

In this function, replace the variables x,y, and z by

$$x = \frac{x_l}{focal}z$$

$$y = \frac{y_l}{focal}z$$

$$z = z$$

and define

$$v_{1z}(z) = v_1\left(\frac{x_l}{focal}z,\ \frac{y_l}{focal}z,\ z\right) \tag{3}$$

$$v_{2z}(z) = v_2\left(\frac{x_l}{focal}z,\ \frac{y_l}{focal}z,\ z\right) \tag{4}$$

and

$$r_z(z) = \frac{1}{R}\sqrt{v_{1z}^2(z) - v_{2z}^2(z)} \tag{5}$$

Gives

$$\begin{aligned} f_z(z) &= f\left(\frac{x_l}{focal}z,\ \frac{y_l}{focal}z,\ z\right) \\ &= \sum_i v_{2zi}(z)w(r_{zi}(z)) \end{aligned} \tag{6}$$

The function (6) then determines density values at any given depth z along a particular viewing ray. Figure 4.2 shows a typical example of density curve as the function of depth z along a viewing ray. An overall blending density curve is just the summation of individual density curves. Figure 4.3 illustrates a blending density curve combined from three individual density curves. The intersection point between a viewing ray and a blending surface is found at depth z_s at which the blending density curve crosses the z-axis.

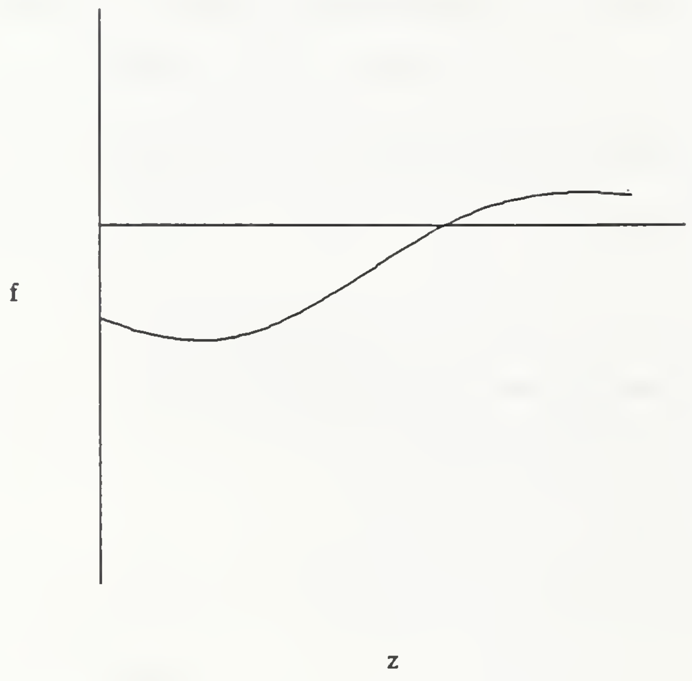


Figure 4.2

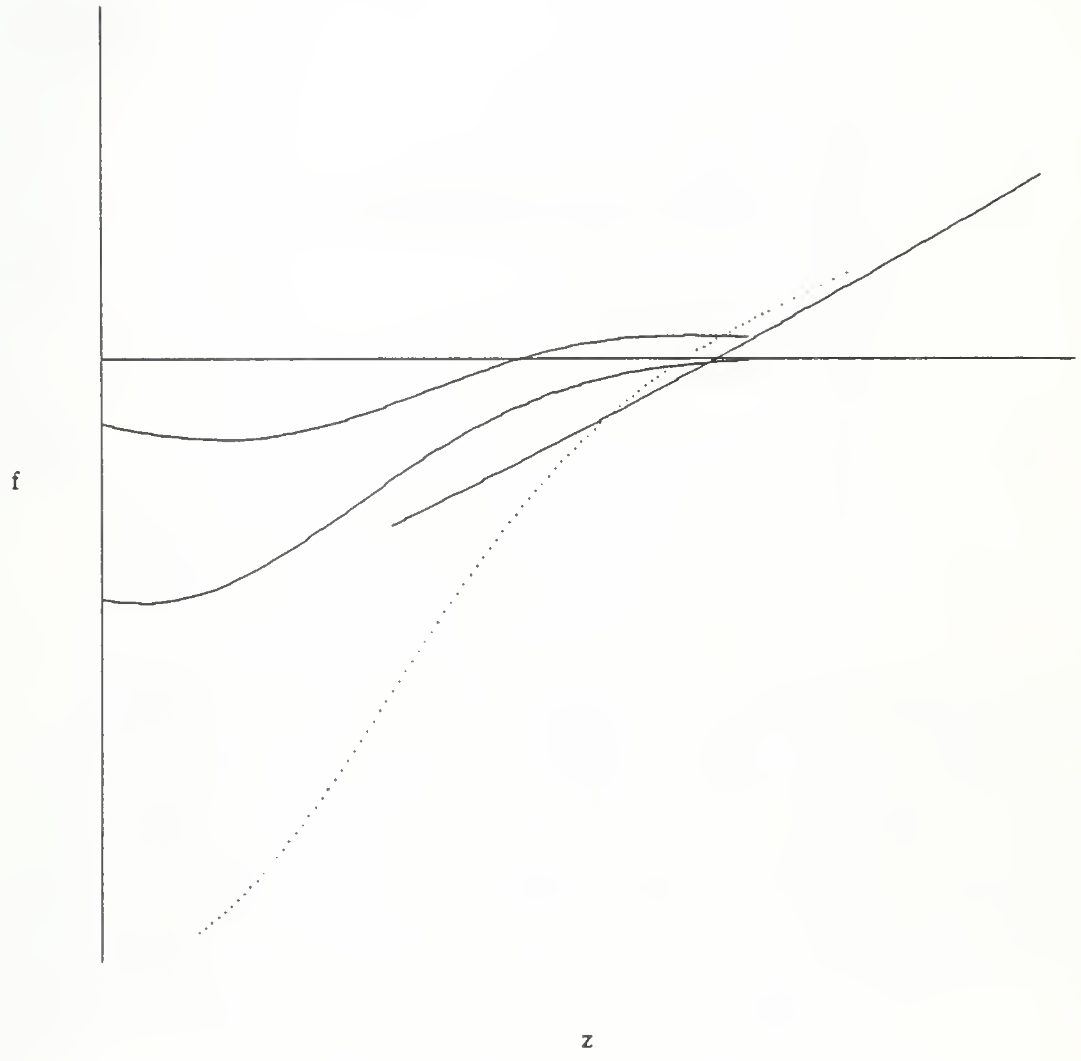


Figure 4.3

That is, the blending surface is determined by the conditions

(1) $f(\overline{P}) = 0$;

(2) $|\nabla f| > \epsilon$, where $\epsilon > 0$, a small positive number;

(3) $f(\overline{P} + \nabla f \delta) > 0$ and $f(\overline{P} - \nabla f \delta) < 0$, for $\delta > 0$, also a small positive number;

(4) For some i, $r_i \leq 1$.

If there is only one disk, then the surface we seek is just the disk itself.

If there is more then one disk, an analytic solution is not feasible and we must proceed numerically, using, for example, Newton iteration.

The computational efficiency of the necessary numerical procedures (such as exponential function and square-root function) can be improved by table look up techniques and linear interpolations.

Newton iteration converges rapidly, but requires an initial guess close to the solution. The method discussed below caused all Newton iterations to converge in our experiments.

This initial guess of z along a particular viewing ray can be made as follows. Assume a ray l intersects with n disks. Let z_i be the depth of the intersection point between ray l and disk i. Take the initial guess z_0 by

$$z_0 = \frac{1}{n}\sum_{i=1}^{n} z_i. \tag{7}$$

This simple method for guessing z's has proved to be very effective by our experiments. A better guess can be given by taking the weighted sum of z_i's used in the rendering algorithm been discussed below.

If neighboring surface points have similar depth z, spatial coherence can be used to find the initial guess effectively, one need only guess the depth of the current point to be that of the last point processed. This ordinarily makes iterations converged extremely fast. This spatial coherence assumption will be true except at points of surface discontinuity. With this spatial coherence assumption, our experiments show that the number of Newton iterations required per point average about 2.5.

4.2. Shading

Once the surface point, surface normal, and surface reflectance properties have been defined at a pixel, an illumination model must be applied to produce the final color at that pixel. A number of models have been developed for this purpose (*Blinn*[17], *Phong*[76]). The lighting model in this research is the one largely simplified from Phong's model, since this is sufficient to show the shapes of our surfaces.

In our procedure, the color at a pixel is given by

$$C = C_f(\vec{L}\cdot\vec{M})^2 \tag{8}$$

where C_f is a reflectance color. $\vec{L}$ is light source direction, and $\vec{M}$ is the surface normal.

To find surface normal at each surface point, we proceed as follows. Once a z solution is found at a particular pixel, it is substituted into eq. (2) to get the x and y coordinates of the same surface point in viewing space. The surface normal is then found by taking the gradient of the surface defining function, f.

$$\bar{n} = \nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}) \tag{9}$$

For the function defined in our model this is readily done, namely

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x}[\sum_i v_{2i} w(r_i)]$$

$$= \sum_i [\frac{\partial v_{2i}}{\partial x} w(r_i) + v_{2i}\frac{\partial w(r_i)}{\partial x}] \tag{10}$$

$$\frac{\partial f}{\partial y} = \sum_i [\frac{\partial v_{2i}}{\partial y} w(r_i) + v_{2i}\frac{\partial w(r_i)}{\partial y}] \tag{11}$$

and

$$\frac{\partial f}{\partial z} = \sum_i [\frac{\partial v_{2i}}{\partial z} w(r_i) + v_{2i}\frac{\partial w(r_i)}{\partial z}] \tag{12}$$

which can be evaluated explicitly for each of the produced distributions.

4.3. A Fast Approximate Rendering Algorithm

We just described a basic rendering algorithm for view independent blending surfaces. However, in many applications rendering speed is important while a certain amount of small error in the generated surfaces are tolerable. For use in such situations, we now describe an approximate rendering algorithm based on the view dependent model introduced in Section 4.2.8.

Recall the definition (21) from Chapter 4 of blending function.

$$z = \frac{\sum_i z_{pi} w(r_{pi})}{\sum_i w(r_{pi})}$$

where z_{pi} is the z depth of the intersection point, pi between a viewing ray and disk i, and $w(r_{pi})$ is weight at point pi on disk i. All that this formula requires is to calculate the intersection point between a given viewing ray and each disk it intersects. These calculations are very simple and fast.

Specifically, A viewing ray through a particular pixel in the image plane is given by

$$(x,y,z) = (\frac{x_I}{focal}z,\ \frac{y_I}{focal}z,\ z) \tag{13}$$

A plane passing through a disk is defined by

$$Ax + By + Cz + D = 0 \tag{14}$$

where

$A = n_x,\ B = n_y,\ C = n_z,$

$D = -(n_x x_0 + n_y y_0 + n_z z_0),$

and

(n_x, n_y, n_z) is the surface normal of the given disk,

(x_0, y_0, z_0) is the center of the disk.

A intersection point must be on both the viewing ray and the disk. So substitute (13) into (14), we have

$$A\frac{x_l}{focal}z + B\frac{y_l}{focal}z + Cz + D = 0$$

Solve this equation for parameter z, we get

$$z = -\frac{D}{A\frac{x_l}{focal} + B\frac{y_l}{focal} + z} \tag{15}$$

Given this z, then x and y are get from (13).

After we found an intersection point, we have to check if this point is on the given disk, i.e. to see if

$$r = \frac{1}{R}\sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2} \leq 1 \tag{16}$$

or equivalently

$$(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 \leq R^2 \tag{17}$$

If the condition (17) is true, we take this disk into account.

Shading can be done using an approximation method as well. First we shade each contributing point according to surface normal of corresponding disk respectively. Then we compute a weighted sum of shading values from all contributing points and normalize the weighted sum, yielding an final approximation shading value.

4.4. Fast Hidden-surface Processing

The following technique can be used to speed up hidden surface process. All disks are divided into subgroups according to disks' orientations. Group all disks whose normal vectors have same signs for all its three components, n_x, n_y, and n_z, together. This subdivision which divide orientation space into eight octants is depicted in Figure 4.4. A normal vector must lie in one of eight octants according to the signs of n_x, n_y, and n_z.

When a camera is placed in such a way that the line of sight of the camera is parallel to one of axes of the orientation coordinate system, then one of subdivision planes is perpendicular to the camera direction. In such a case four groups behind this plane can be skipped during rendering, since all normal vectors in these partitions are facing away from the camera. However, in most of cases a camera is viewed from an arbitrary direction, so that there will be only one octant which can be skipped.

Finer subdivision of the orientation space can be introduced in order to skip as many invisible disks as possible. However, any finite subdivision of the orientation space would not classify all invisible disks for an arbitrary viewing direction. A simple test for each disk in the rest of subgroups can still cut off a certain amount of disks. Let $\vec{l}$ be a viewing ray (Figure 4.5).

$$\vec{l} \cdot \vec{n} = |\vec{l}||\vec{n}|\cos(\theta) \tag{39}$$

From this, we have

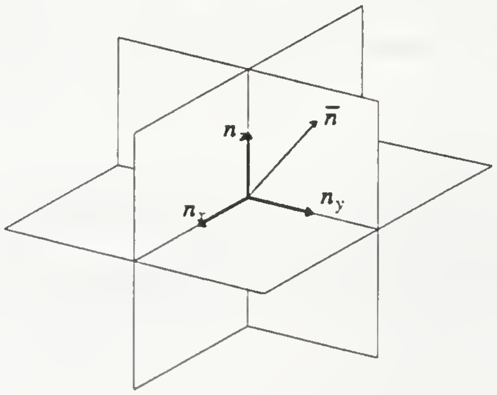


Figure 4.4

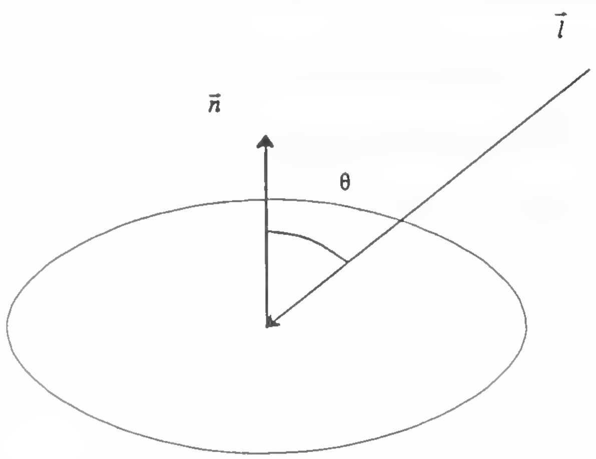


Figure 4.5

$$\cos(\theta) = \frac{\vec{l}\cdot\vec{n}}{|\vec{l}\,||\vec{n}\,|} \tag{40}$$

So

$$\cos(\theta)\geq 0 \quad \textit{iff} \quad \vec{l}\cdot\vec{n}\geq 0 \quad \textit{iff} \quad 0\leq\theta\leq\frac{\pi}{2;}$$

and

$$\cos(\theta)<0 \quad \textit{iff} \quad \vec{l}\cdot\vec{n}<0 \quad \textit{iff} \quad \frac{\pi}{2}<\theta<\pi.$$

Therefore, $\vec{l}\cdot\vec{n}$ tells us whether or not a disk is facing away from the viewing ray $\vec{l}$. If $\vec{l}$ and $\vec{n}$ both are unit vectors, then $\cos(\theta) = \vec{l}\cdot\vec{n}$. A disk can be ignored in rendering if $\cos(\theta)\leq -\epsilon$, where ϵ is a small positive number. The calculation of $\cos(\theta)$ requires only one dot-product operation.

So about one half of disks can be eliminated before actually rendering them. This hidden-surface removal process improves the efficiency of rendering program by almost a factor of two.

4.5. A Scan-Line Algorithm

We can readily develop a scan-line algorithm which is analogous to that of maintaining a list of potentially visible polygons in more conventional polygon-rendering algorithms. This algorithm consists of three major steps:

(1) Eliminating

(2) Y-sorting

(3) Scanning

Eliminating

This first step is to eliminate those subgroups of disks whose orientations are away from the camera as described in the last section.

Y-sorting

For each potentially visible disk, we first project perspectively its center and its radius onto the image plane, gives (x_I, y_I) and R_I respectively. This defines a rectangular box in image plane having (x_I, y_I) as its center and $2R_IC_R$ as its edge length, where $C_R>1$ is a constant, and four corners defined by four values: *xmin*, *ymin*, *xmax*, and *ymax*. The effect of multiplying by C_R is to scale the box by a certain ratio.

A bucket-sort on *ymin* is then performed. An array, named *scan_table[0..MAX_ROW-1]*, is created. An entry of this table, *scan_table[y]*, $0 \le y < MAX\_ROW$, is a ponter to a list of disk records the *ymin*'s of which are equal to *y*. Within each bucket, all records are further sorted on *xmin*.

Scanning

Following procedure describes the scanning process:

```
for (y=0 to MAX_ROW-1) {
    if (scan_table[y] is not empty)
        move all records from scan_table[y] to y-active-list;
    for (x=0 to MAX_COLUMN-1) {
        copy all records in y-active-list to x-active-list,
          for which xmin≤x;
        evaluate the blending surface point at pixel x,y);
        remove all records from x-active-list, for which xmax≤x;
    }
    remove all records from y-active-list, for which ymax≤y;
}
```

During scanning process, a *y-active-list* is maintained to contain all disk records which are affecting the current scanline as follows.

At the beginning of new scan-line *y*, if the bucket *scan_table*[*y*] is not empty, then all records in this bucket are moved from the bucket to the *y-active-list*. All records in the *y-active-list* are kept sorted on *xmin* all the time.

At the end of a scan line *y*, all records in the *y-active-list* are checked against their *ymaxs*. If someone's *ymax* is equal to or great than *y*, then this record is terminated. Hence it is removed from *y-active-list*.

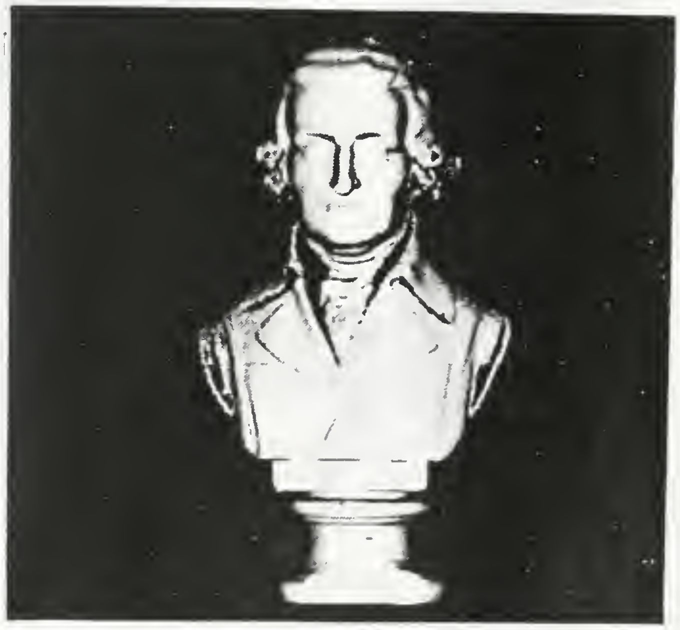
Similarly, an *x-active-list* is used to maintain all disk records which are affecting the currently scanned pixel. This is done in a manner exactly analogous to that of the *y-active-list*, except that a record is not moved but copied to *x-active-list*.

Within *x-active-list*, boxes are sorted on *z* depth. A sequence of disks which are close together are formed into a subgroup. A blending operation will be performed only within each subgroup.

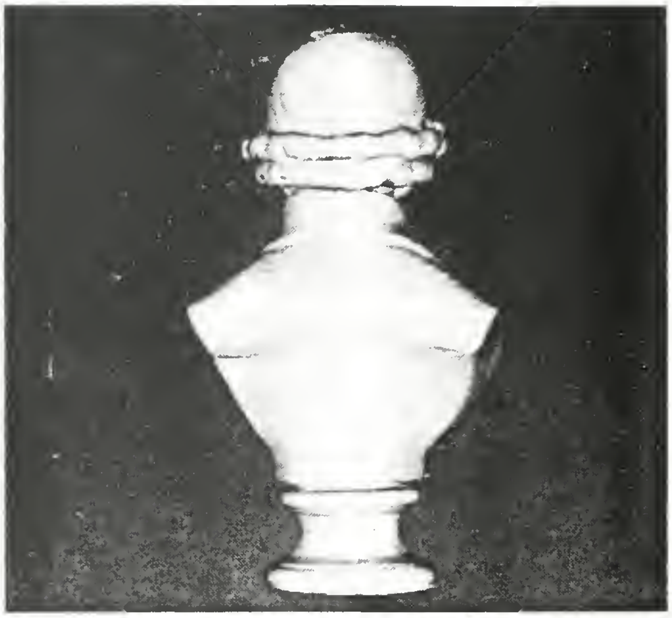
At any point in this procedure, we have a list of all disks which are potentially affecting the blending surface at the current pixel. Thus the algorithms presented previously for both view independent and view dependent models can be applied for evaluating a surface point and shading such a surface point. This is done by considering successive subgroups in the $x - active - list$.

4.6. Some Experimental Results

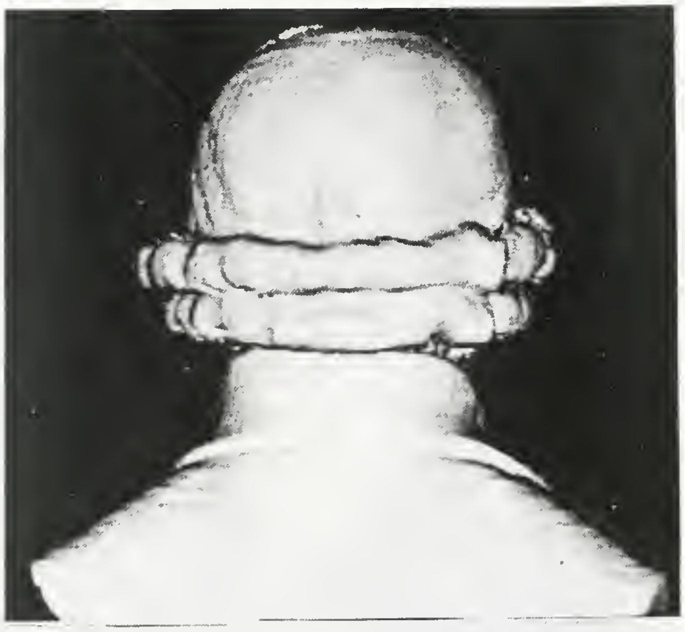
Figure 4.6 (a), (b), (c), and (d) are generated from our three-dimensional disk model representations of the Haydn bust.



(a)


(b)

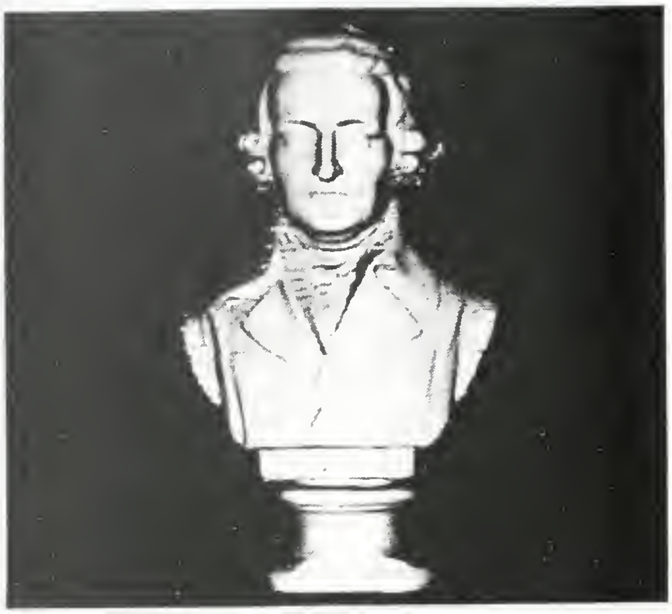
(c)



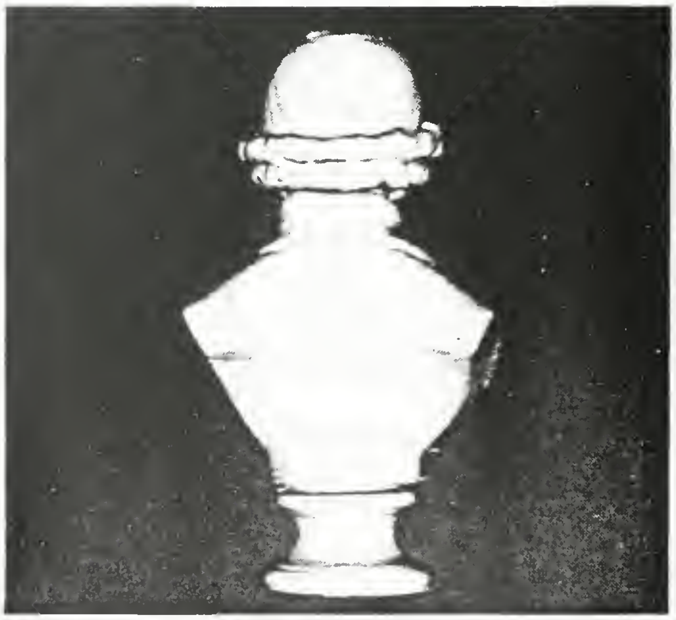
(d)

Figure 4.6

There are a number of gaps appearing in the images which are caused by the shadow problem with the three-dimensional sensing device used (see the description of the device in Appendix). Although this problem is not an emphasis of this thesis, we have tried to make up these gaps for nicer pictures. There are two possible approaches to this problem. One is to interpolate the gaps by some numerical methods. Another is to improve three-dimensional sensing techniques to provide more complete data. Figure 4.7 (a) and (b) show the results from our interpolation method. Here we first use Bezier bicubic parametric surface patches to interpolate these gaps, then the disk representations are constructed to approximate such Bezier bicubic parametric surface patches. The difficulty of interpolation for the gaps shown in Figure 4.6 is coming from the highly irregularly shaped boundaries of the gaps in three-dimensional space. The interpolation techniques for this kind of gaps are interesting. Therefore, in general an interpolation technique would not be able to recover the surface details of the missing portions. The better solution is the second.



(a)

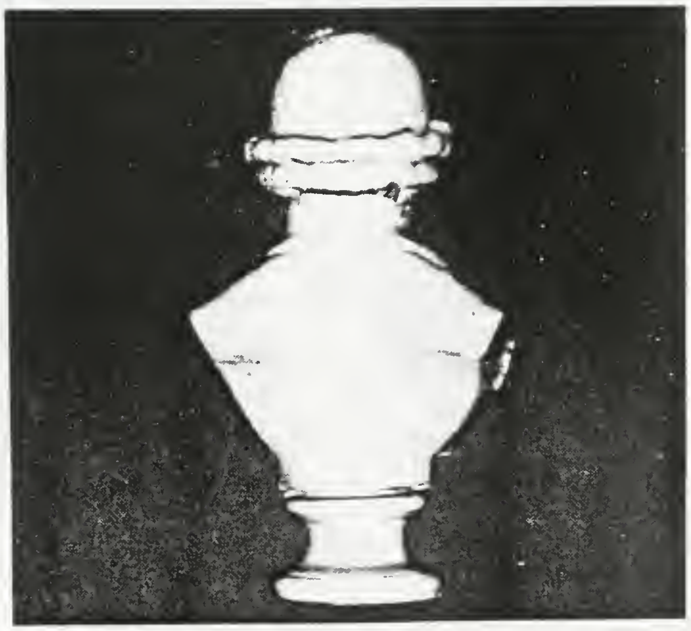


(b)

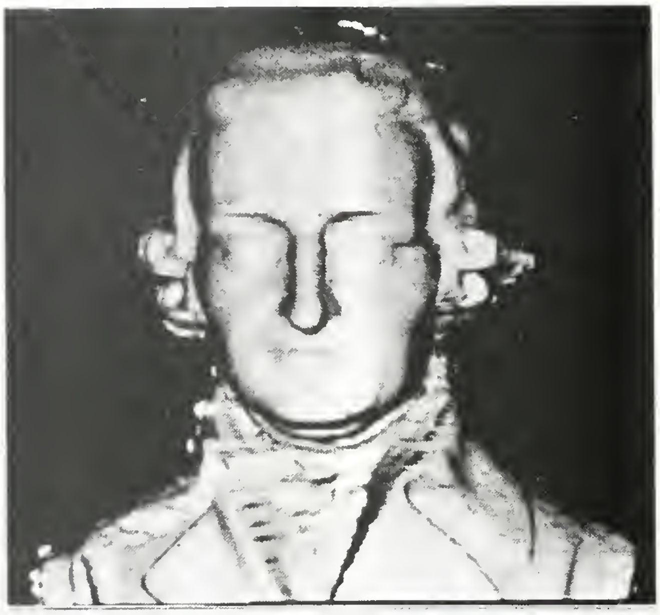
Figure 4.7

Figure 4.8 (a), (b), (c), (d), (e), (f), (g) and (h) are generated by a disk model representation of the resolution four times lower than that of Figure 4.7.

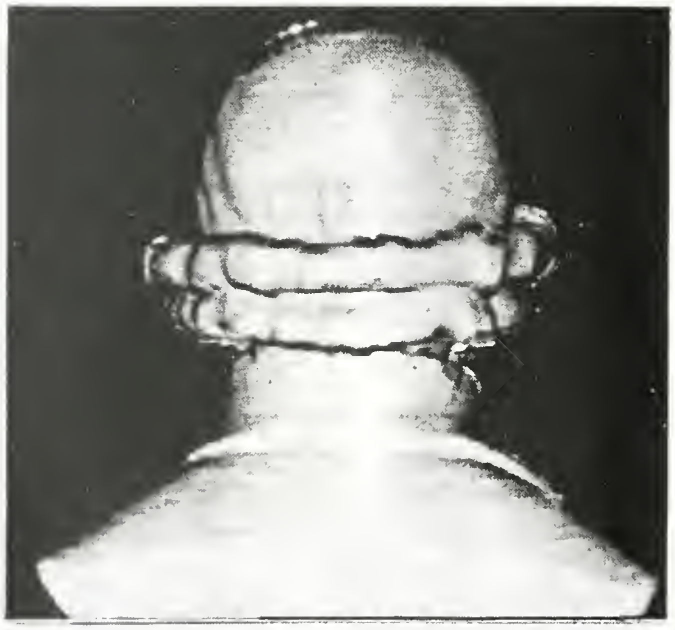
(a)



(b)



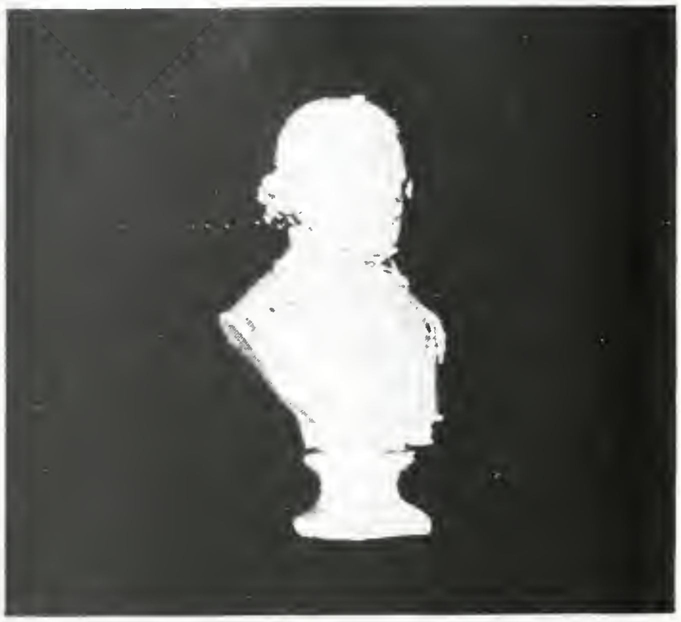
(c)



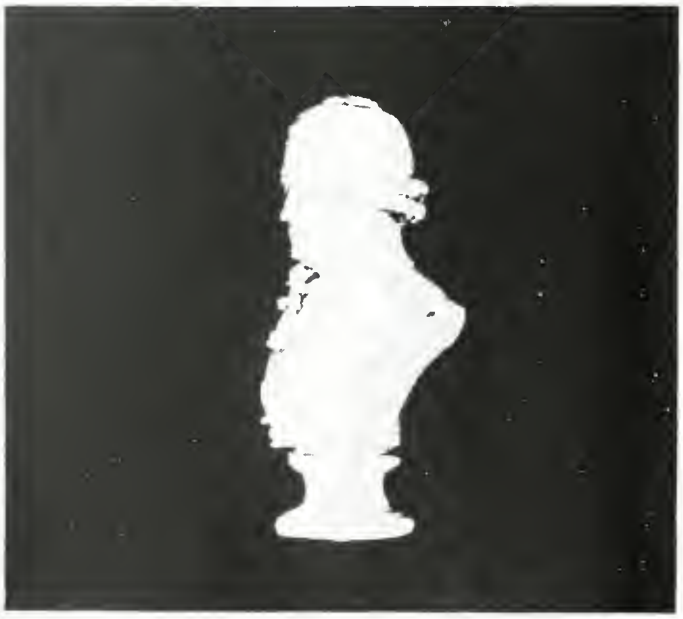
(d)

(e)

(f)



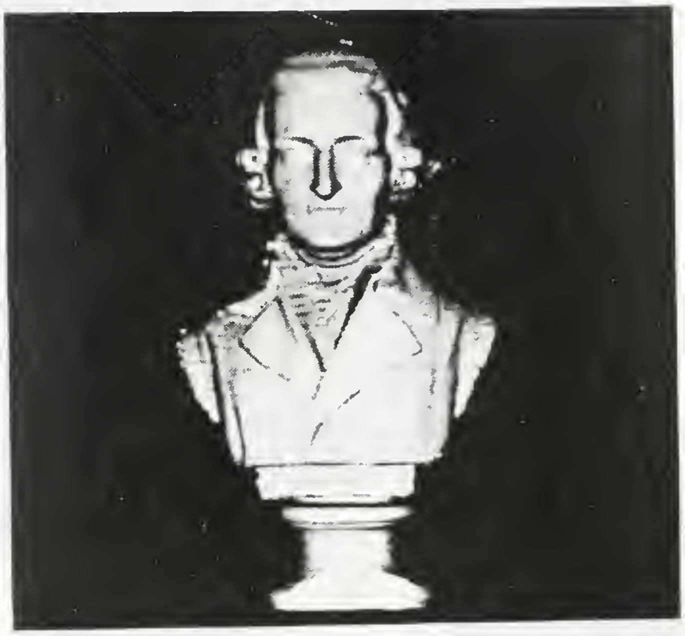
(g)



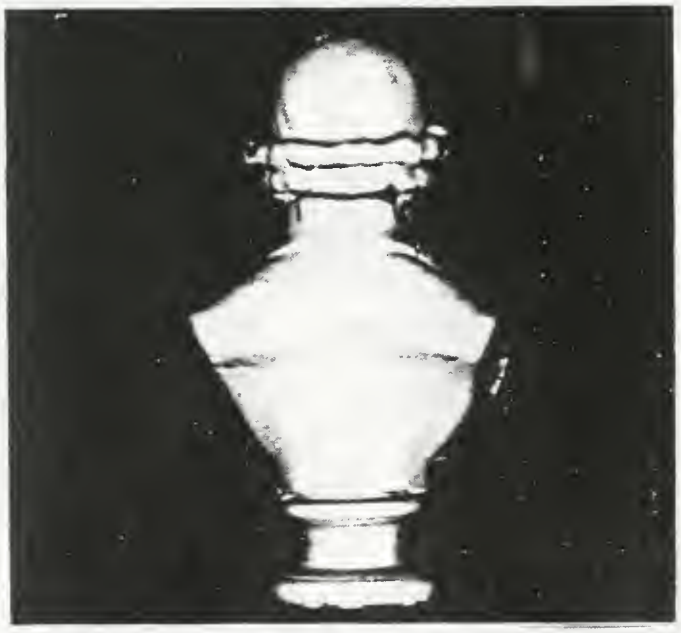
(h)

Figure 4.8

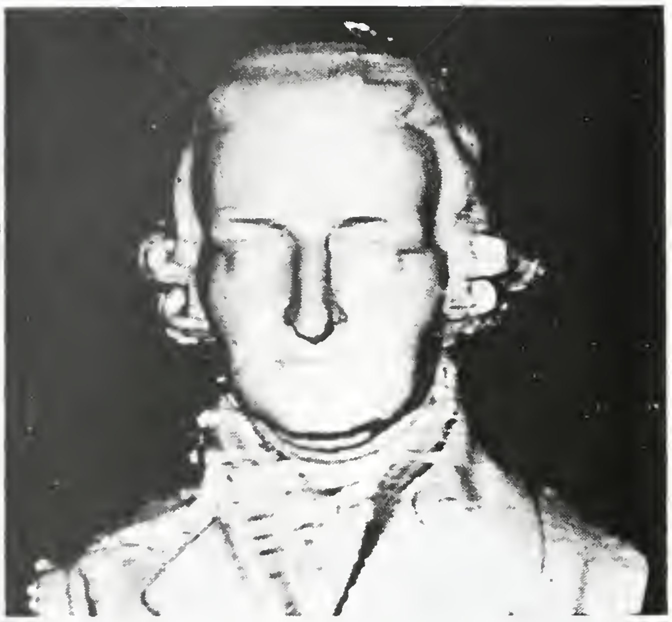
Figure 4.9 (a), (b), (c) and (d) are generated by a multi-resolution model representation with three levels. The total number of disks for these pictures are same as that of Figure 4.8 and the size of disks of the middle level is same as that of Figure 4.S. By comparing Figure



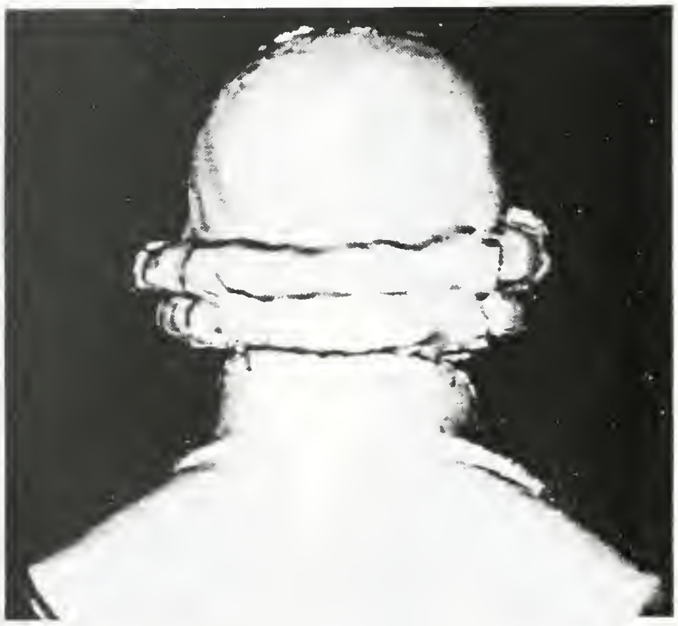
(a)



(b)



(c)

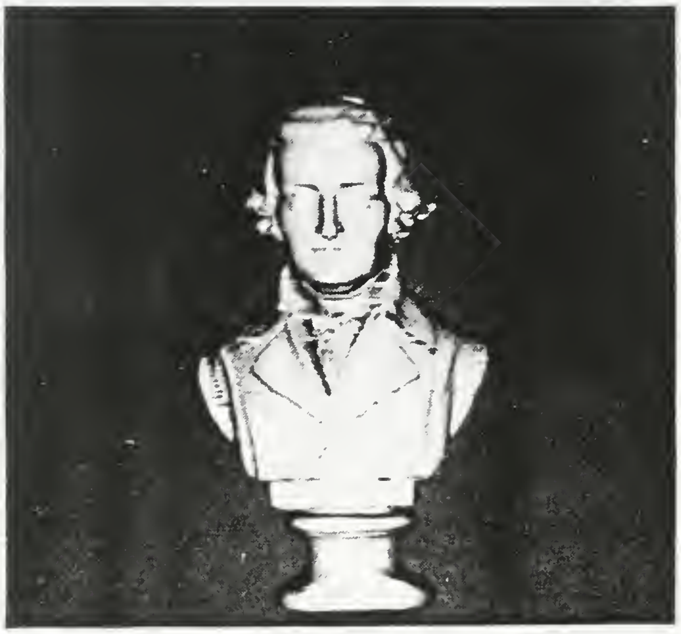


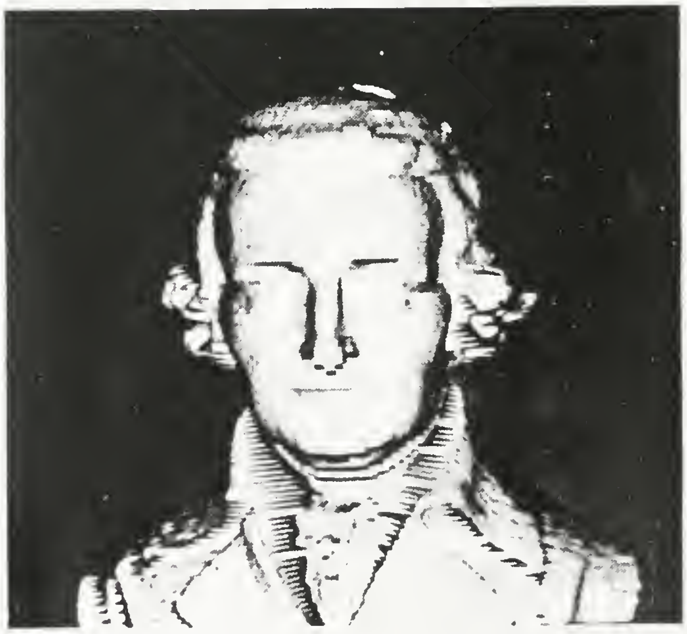
(d)

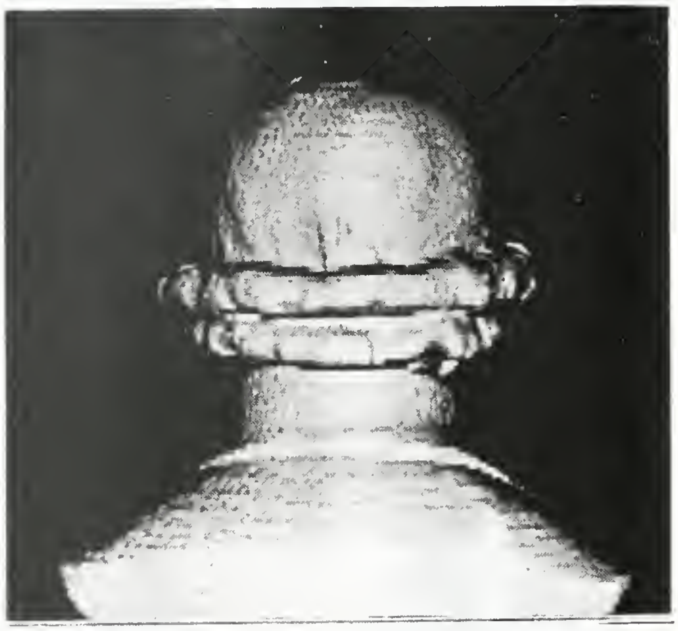
Figure 4.9

4.8 (c) Figure 4.9 (c), one can see that some surface details, such as that in the area of front neck tie, are not shown out in Figure 4.8 (c), but are visible in Figure 4.9 (c). This is because of that by using multi-resolution structure some surface details has been refined adaptively by finer disks.

Figure 4.10 (a), (b) and (c) are generated from polygonal model by using the same


(a)


(b)



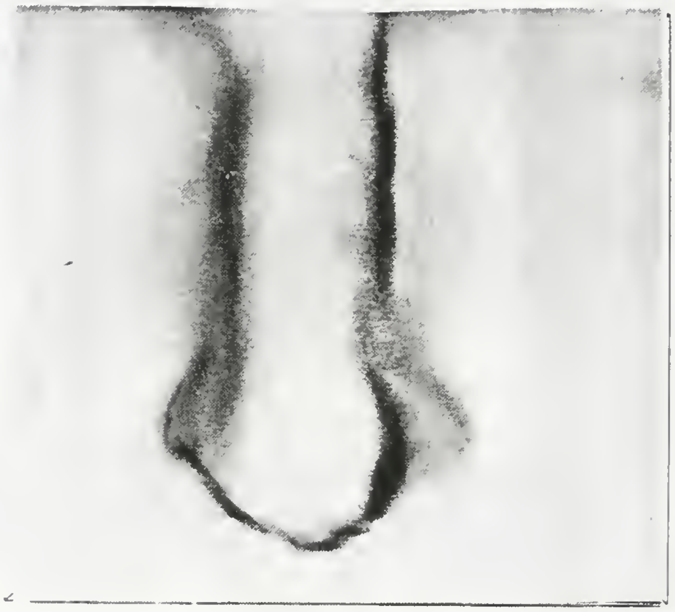
(c)

Figure 4.10

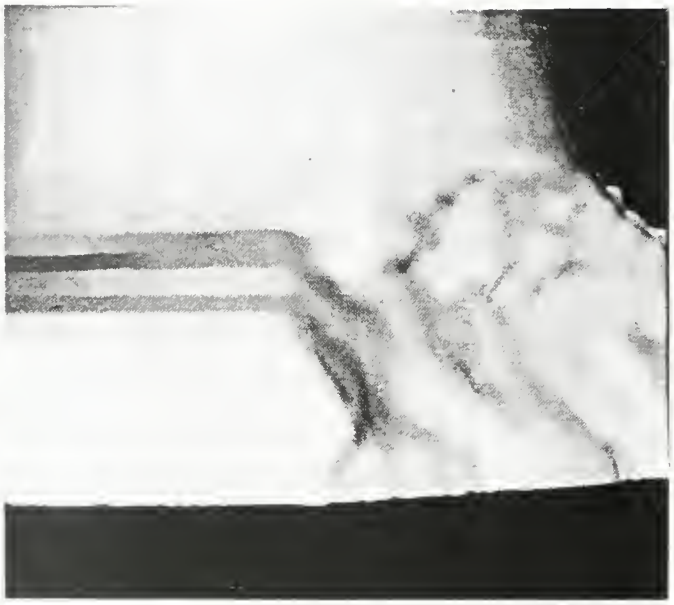
number of surface patches as that of Figure 4.8 and 4.9. Figure 4.10 show some planar facets clearly. Better smoothed shading can be obtained using Gouraud's surface normal interpolation method across the polygons (*Gouraud*[40]). This technique makes surfaces appeared to be smoothly curved, but the occluding boundaries will be still polygonal and if the surfaces intersect, the intersection lines will be also polygonal. However, our blending method generates truly smoothed surfaces. Figure 4.11 are close-ups of different part of the surface.

In some of previous pictures, one see that there are aliasing problems at the occluding boundaries. This is caused by our fast approximation rendering algorithm. Anti-aliasing at the occluding boundaries with our model for fast rendering is interesting and useful. We leave this problem for future research.

Finally, we demonstrate some pictures which are generated by composing the Haydn bust of disk model and some synthetic objects of polygonal model.

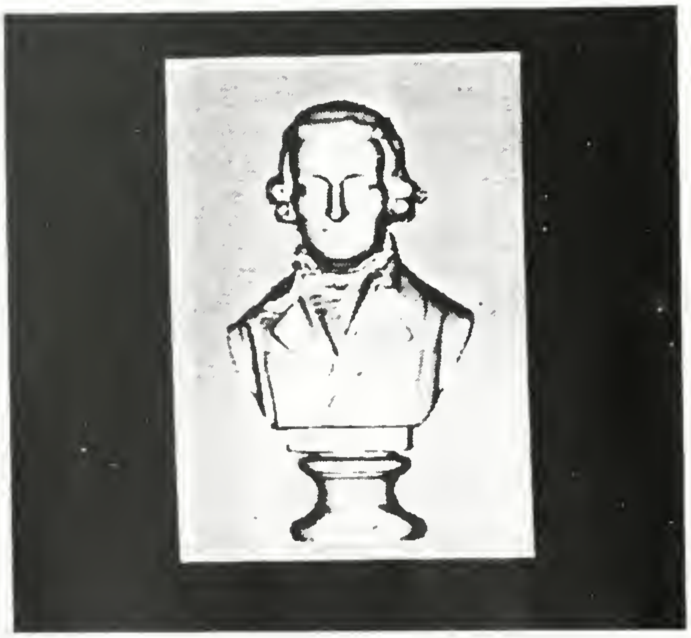

(a) Nose

(b) Hair

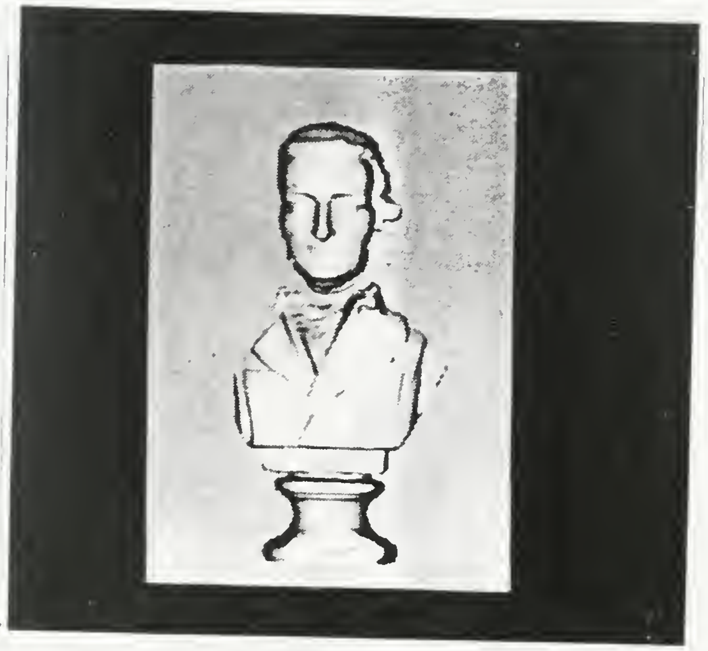


(c) Bottom

Figure 4.11



(a)


(b)

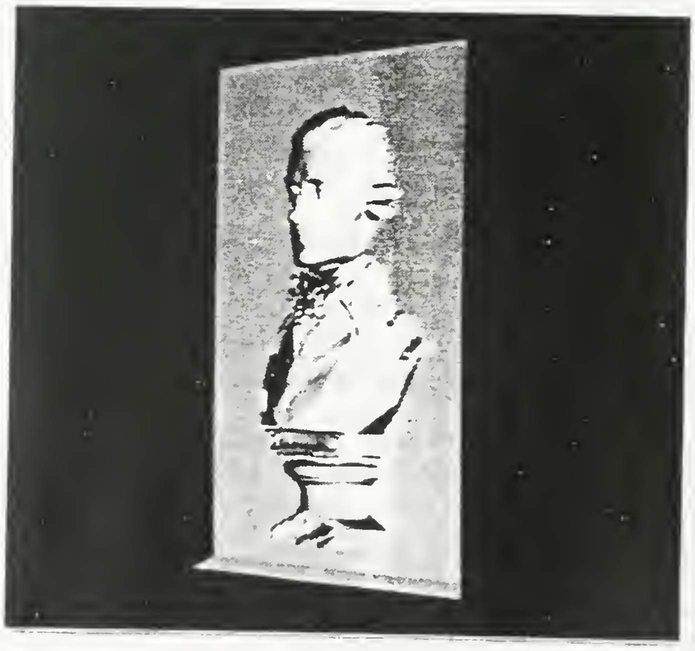

(c)

Figure 4.12

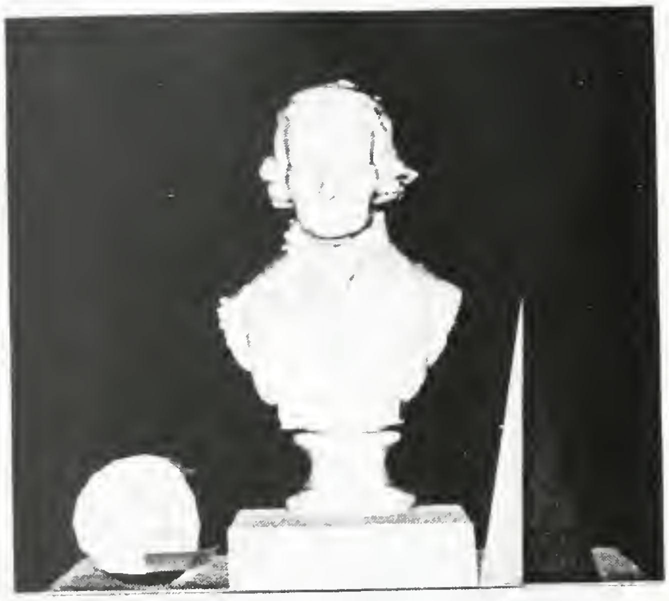


Figure 4.13

In Figure 4.12 (a), (b) and (c), the Haydn bust intersects with a frame. Figure 4.12 (b) is generated by moving the frame forward.

Figure 4.13 shows a set of synthetic objects represented by polygonal model composed with the Haydn bust of fuzzy disk model.

4.7. Efficient Shadow Penumbra Approximation

A modified Z-buffer shadow algorithm is developed for efficient shadow penumbra approximation which can be easily included into Z-buffer based renderers. The algorithm is a largely simplified cone tracing technique using a modified Z-buffer data structure.

4.7.1. Introduction

Shadow casting techniques have received great attention from computer graphics researchers because shadows improve comprehension and enhance the realism of computer-synthesized images. Shadows provide valuable cues to positional relations among light sources and objects appearing in an image, and when used in scene shading, greatly enhance perception of the shape of objects shown in an image.

Max[64] has classified shadow algorithms into five basic types: Z-buffer, area subdivision, shadow volume, preprocessing, and ray tracing. Much previous work has been restricted to point light sources (*Crow*[26]). In this special context, shadowing means classification of each surface point as being in-shadow or not-in-shadow. However, in the real world, light sources are rarely volumeless points. Accordingly, low pass filter techniques (*Reeves*[82]) have been used to soften the edge of shadows to resemble true penumbras, but this is far from satisfactory in most situations. More sophisticated shadow algorithms considering both umbra and penumbra have been generated by ray tracing.

The Z-buffer shadow algorithm (*Williams*[97]) is used very widely due to its simplicity of structure, its ability to comprise the visibility section of most kind of renderers, and its suitability for hardware implementation. This algorithm computes the light-source and viewpoint images by depth buffer algorithms. Each pixel in the viewpoint image is transformed to the light-source image and compared with the depth in the light-source image. If the visible point at a pixel of the viewpoint image is farther from the light than the corresponding depth in the light-source image, it is in shadow. Aliasing problems connected with this algorithm has been reduced by various sampling and filtering techniques (*Reeves*[82]). However, since the light-source in the Z-buffer algorithm is a point source, it is not applicable to penumbra generation.

Ray tracing algorithms generate excellent realistic images, but tend to be very expensive computationally. Simple ray tracing has aliasing problems. Amanatides[2] presented a cone tracing technique for anti-aliasing, and for generation of fuzzy shadows and dull reflections. To improve the efficiency of ray tracing, Kay & Kajiya[52] and Rubin & Whitted[87] created a hierarchy of bounding volumes to reduce the number of bounding volume intersection checks; Glassner[38] and Kaplan[51] employed three-dimensional space subdivision in culling objects; Heckbert & Hanrahan[41], Kaplan[51], and Shinya, et al.[91] made use of the coherence which exists between similar rays. Although a certain degree of improvement in efficiency has been attained by the above efforts, ray tracing's computational expense still restricts its applications.

We will now introduce a new approximated shadow penumbra method. The new penumbra approximation model utilizes a modified Z-buffer technique, and is quite simple in structure and very efficient in implementation. Since it is an approximation to the correct solutions, a certain amount of error will occur. We analyse various kinds of sources causing the error and present methods to reduce the error as much as possible.

4.7.2. Penumbra Approximation Model

The intensity value at a surface point in a given environment viewed from a certain direction is determined by emitted light, reflected light, and refracted light from that point. The most accurate approach to this problem known to the author is the radiosity method (*Cohen*[22]), which accounts for all interreflections of light between diffuse and non-diffuse surfaces in a given environment. The high quality of the images it generates is paid for by the extremely large computing cost of both the preprocessing and the final rendering processes. Rather than accounting for complex interreflections within the environment as does the radiosity method, ray tracing techniques consider primarily secondary reflected and transmitted rays from mirror and refracted directions. Although ray tracing techniques are not as accurate as radiosity methods, they have been able to produce strikingly realistic images in less computing time. However, in this section our interest is restricted to the shadow casting problem. We proceed by first looking at the approach taken to this problem by previous ray tracing techniques.

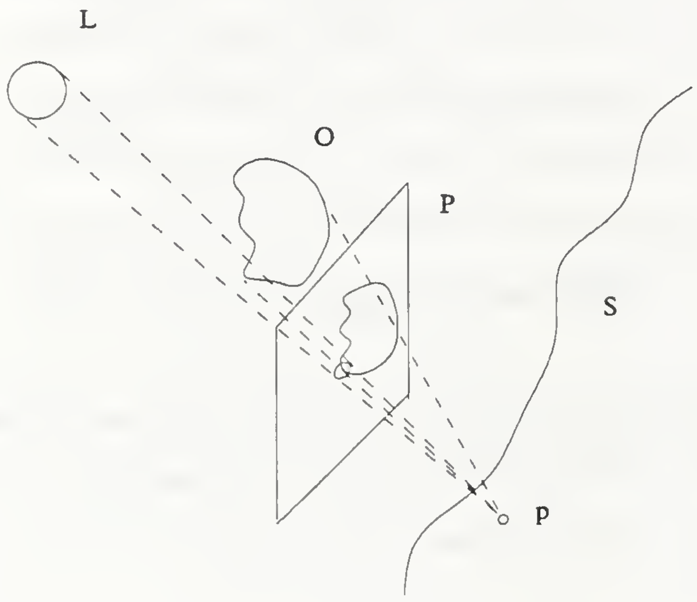


Figure 4.14. A simplified cone tracing scheme

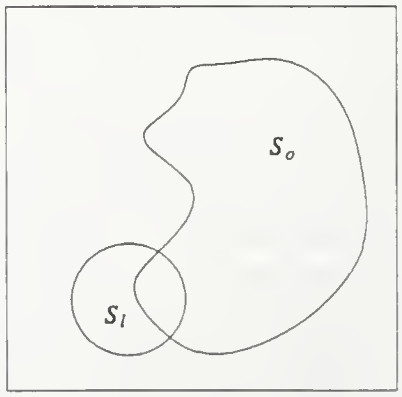


Figure 4.15. The projections of object and light source

As shown in Figure 4.14, the obscuring object O we wish to consider is placed between the light source L and the surface S, generating a shadow on the surface S. To determine the illumination at a point p on the surface S we can place a projection plane in space with the point p as the center of perspective projection. Both object and light source are then projected perspectively onto the projection plane. Let S_l and S_o be the projected areas on the projection plane of the light source and of the object, respectively (Figure 4.15). There are three possible cases. If $S_l \cap S_o = NULL$, then the surface point p is fully illuminated. If $S_l \subseteq S_o$, i.e. S_l is completely covered by S_o, then the surface point p is in full umbra. Finally, if $S_l \cap S_o \neq NULL$ and $S_l \cap S_o \neq S_l$, i.e. S_l is partially covered by S_o, then the surface point p is in penumbra. Thus the illumination level at the surface point p can be taken to be proportional to $1-\frac{|S_l \cap S_o|}{|S_l|}$, where $|A|$ denotes the area of the set A.

In a complex scene, we have several objects in the space. A general solution to compute the portion of the light source visible from any single surface point would be some kind of hidden surface removal algorithm, using the surface point as the view origin. This is a form of cone tracing.

Obviously this method to compute the penumbra at each visible surface point in the scene would be extremely expensive. We now propose a similar but far less expensive approximation to this approach. The scheme shown in Figure 4.16 is same as the one shown in Figure 4.14 except that the shadow mask is projected onto the projection plane by taking the center of the light source as the projection center. Instead of doing projections for objects in the scene from every point on the surface S at which the shadow value is to be computed, we propose using a single projection for objects from the center of light source as in a typical Z-buffer algorithm, and then using this shadow mask to accomplish cone tracing from the light source for all points on S. This eliminates a large amount of computation. But how accurate

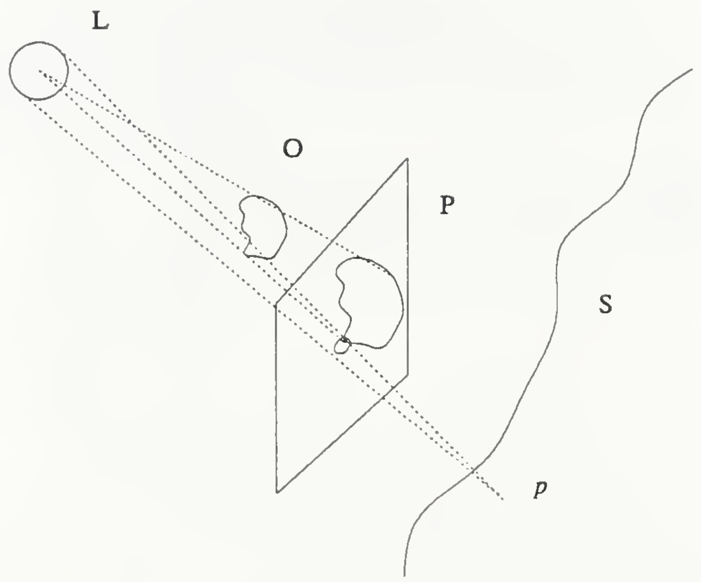


Figure 4.16 The projection center for objects is the center of the light source.

is the shadow generated by this method? To answer this question, we simplify the case to two dimensions as in Figure 4.17, and then apply our conclusions to the three dimensional case.

4.7.3. Error Estimation

In Figure 4.17, the light segment L of length 2^*R casts the shadow of the object O onto the surface S. Points on S between u_1 and u_2 are in full umbra. Points from u_1 to p_1 and from u_2 to p_2 are in penumbra. All points beyond p_1 or p_2 are fully illuminated. A projection line P is placed in space parallel to L. We take the center of L, O_l as the projection center. The segment P_o on P is the projection of object O. Let us look at the lower portion of S. The correct bound between umbra and penumbra should be at u_1, the intersection point of line l_1 and S. Therefore the practical bound we can get by using mask P_o is u_1'. An error ΔS, the shift of the boundary between umbra and penumbra, is introduced due to taking the center of the light source as the projection center to generate the shadow mask. An important

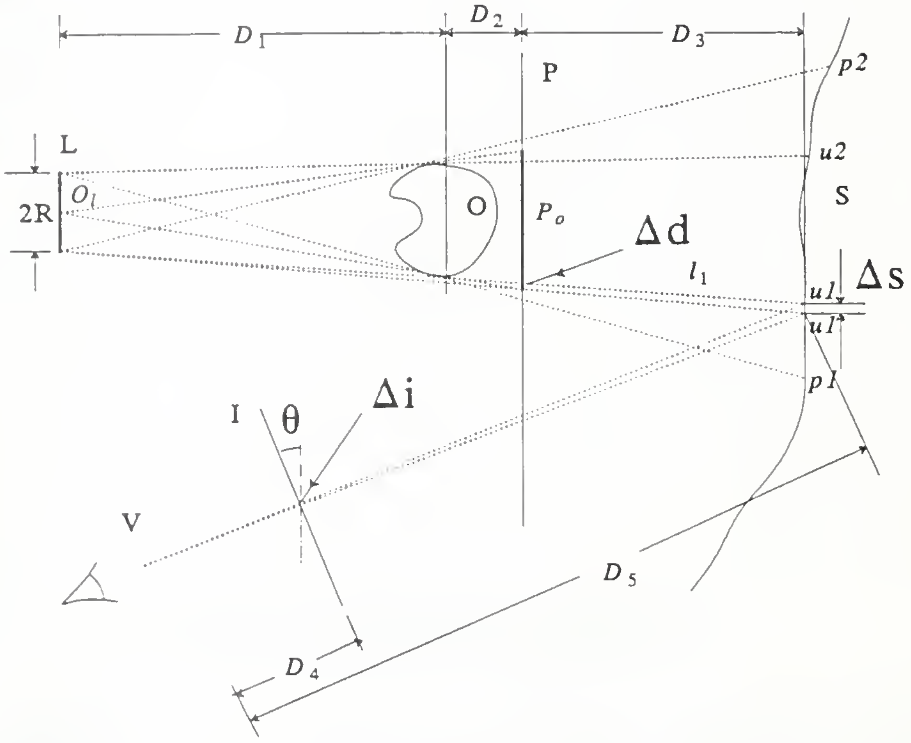


Figure 4.17 Errors introduced by this approximation model

question is: how large is ΔS in a given configuration? we estimate this error.

The occlusion boundary on P is shifted by Δd, which in turn results in the boundary between umbra and penumbra on surface S being shifted by ΔS from u_1 to u_1'. Since

$$\frac{\Delta S}{D_1+D_2+D_3} = \frac{\Delta d}{D_1+D_2} \tag{20}$$

then

$$\Delta S = \Delta d * \frac{D_1+D_2+D_3}{D_1+D_2} \tag{21}$$

Note that

$$\frac{\Delta d}{D_2} = \frac{R}{D_1}, \quad i.e. \quad \Delta d = R*\frac{D_2}{D_1} \tag{22}$$

Substituting (22) into (21), we have

$$\Delta S = \frac{R*D_2*(D_1+D_2+D_3)}{D_1*(D_1+D_2)} \tag{23}$$

Let V be the camera position, and I be the image plane. The error ΔS and the error Δi in the image caused by ΔS have the relationship

$$\frac{\Delta i}{D_4} = \frac{\Delta S*cos(\theta)}{D_5} \tag{24}$$

From (24) we get

$$\begin{aligned}\Delta i &= \Delta S*\frac{D_4}{D_5}*cos(\theta) \\ &= \frac{R*D_2*(D_1+D_2+D_3)*D_4}{D_1*(D_1+D_2)*D_5}*cos(\theta)\end{aligned} \tag{25}$$

When $D_2<<D_1$ and $D_2<<D_3$, (25) can be simplified into

$$\Delta i = \frac{R*D_2*(D_1+D_3)*D_4}{D_1^2*D_5}*cos(\theta) \tag{26}$$

For a given $\epsilon>0$, if we want

$$|\Delta i| = \frac{R*D_2*(D_1+D_3)*D_4}{D_1^2*D_5}*|cos(\theta)| \leq \epsilon$$

we only need

$$\begin{aligned}D_2 &\leq \epsilon*\frac{D_5}{D_4}*\frac{D_1^2}{R*(D_1+D_3)}*|cos(\theta)| \\ &\leq \epsilon*\frac{D_5}{D_4}*\frac{D_1^2}{R*(D_1+D_3)}\end{aligned} \tag{27}$$

Equation (27) suggests that for a given configuration, if D_2 satisfies (27), the error introduced by the above scheme in the final image is less than the given limit ϵ. In other words, there is always a range for D_2 such that we can maintain the results as accurately as required. By doing a similar analysis for the boundary between penumbra and fully illuminated area we can get the same result as the one above. This analysis provides us with an important guideline to designing our algorithm.

Since the shape of an object can be arbitrary and a configuration of light sources and

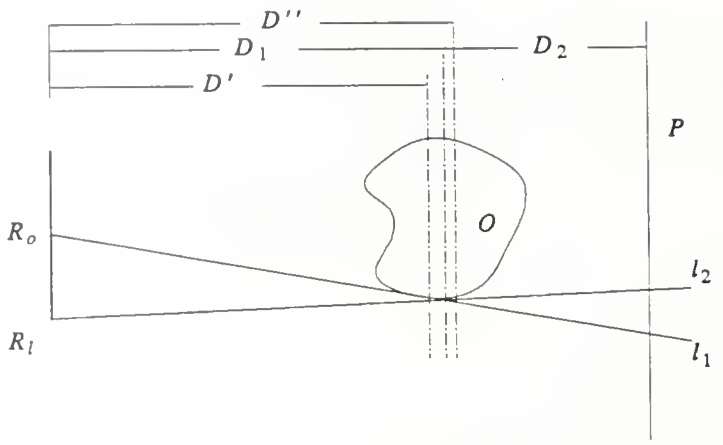


Figure 4.18

objects can also be arbitrary, we have to pay more attention to the errors caused by those conditions. See Figure 4.18. Let l_1 be the line from R_o and tangent to the low boundary of object O at the depth D', and l_2 be the line from R_l and tangent to the low boundary of O at the depth D''. There are many cases that D' will not equal D''. D_1 used in the previous analysis is the depth of the intersection point of l_1 and l_2, and it's often neither equal to D' nor D''. If the depth map is computed from R_o, then D' is known. But both D_1 and D'' are not known. The condition (27) can be applied to minimize the error only if D_1 and D' are close. This assumption is true for many cases. But it will fail in some cases as well. A worst case is shown in Figure 4.19.

There are other kind of errors could be caused by this scheme. For example, in Figure 4.20, there should be a portion of penumbra area on the surface S. But by using the scheme we proposed, this penumbra area will be missing.

4.7.4. Algorithm

We now describe various different implementations of the penumbra approximation

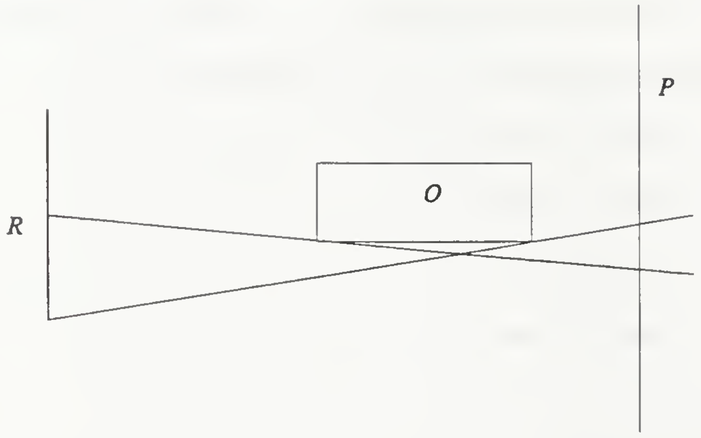


Figure 4.19

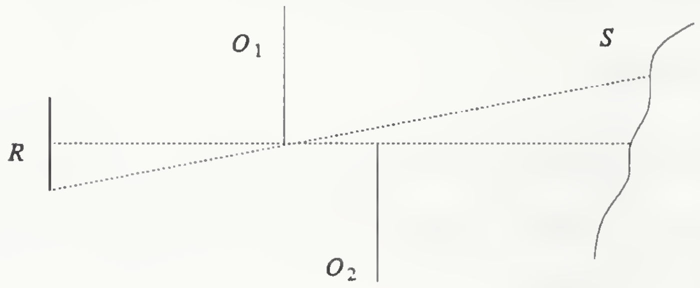


Figure 4.20

method introduced above. The rendering program consists of two passes as in Williams's Z-buffer algorithm. The first pass creates the depth map structures for each light source; and the second pass renders the scene, using such depth map structures to compute shadows.

First Pass: Creating the Depth Maps

Our major goal is to make the shadows generated as accurate as possible without excessive sacrifice efficiency. We would like to develop different implementations with variant accuracy and efficiency to meet the requirements of different applications. We start with the simplest implementation.

(a) A Simple Implementation

As stated in the previous section, minimizing D_2 is the key to maintaining accuracy. But the depths of objects in the scene could vary over such a large range that no single D_2 would satisfy the condition (48). In such a case, if efficiency is critical and a certain degree of degradation in accuracy is tolerable, a naive choice is to place the projection plane in the middle range of the objects' depth. This position can be found efficiently by bounding the objects with a box. First, we find the bounding box for each object; then we find a large box bounding all of these objects' bounding boxes. This large box gives us the range of depth. Let Z_{min} denote the depth value of the front face of the bounding box as viewed from the light source, and Z_{max} the depth value of the back face. We then choose the position to place the projection plane parallel with the front and back faces to be $(Z_{min}+Z_{max})/2$.

(b) **Depth Map Array**

If the error introduced by the above simple approach is intolerable for some application, the scheme can be refined by replacing the simple depth map by an array of depth maps in the range from Z_{min} to Z_{max}.

This array is generated by sequential placement. We start by placing the first projection plane at Z_{min}. Using condition (3), we can find the subrange d_1 associated with the first depth map in which the desired accuracy can be maintained. The next projection plane is placed at $Z_{min}+d_1$. We repeat this displacement until we reach Z_{max}.

We also need to maintain a Z-list, Z[n], for searching the depth map later. The Z-list contains a set of z values stored in increasing order. These z values represent the depth positions at which the depth maps are located.

(c) **Pyramid of Depth Maps**

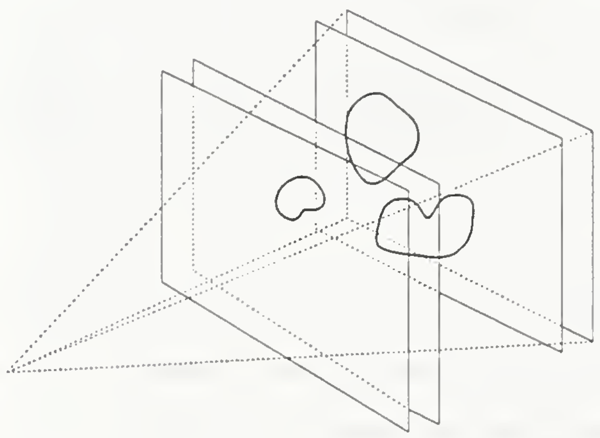


Figure 4.21 A cone includes objects in the scene

If we draw a cone from the center of the projection to include all concerned objects in space (Figure 4.21), we find that for a projection plane located at a certain depth z, any part of the plane outside of the cone is useless for later computations. In other words, the window for a depth map on a projection plane doesn't need to exceed this cone. The closer the projection plane is to the center of projection, the smaller the size of the window associated with the depth map will be. Obviously in the depth map array method, many depth maps have boundaries exceeding the cone, hence a lot of data storage is wasted. This consideration motivates us to develop a pyramidal structure, which fits the cone naturally.

(Pyramidal data structures have been employed in image processing for the purposes of data compression, pattern recognition, etc. However, this technique has also been demonstrated to be useful in computer graphics (*Williams*[98]), for example, to do bandlimited texture mapping, etc.)

Let the depth map at the first level in the pyramid have a size of $n*m$. At the second level we would then have a depth map of $\frac{n}{2}*\frac{m}{2}$, and so on for successive levels. Higher levels can be computed from lower levels by using various kinds of filtering techniques. The simplest

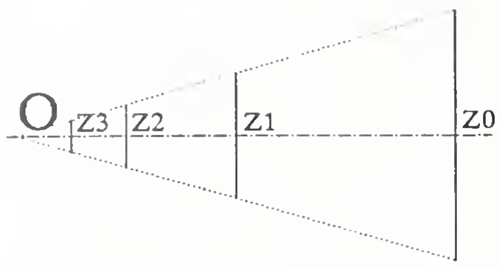


Figure 4.22. Placements of depth maps in pyramidal structure

filter, and the one we implemented in our experiments, is local averaging. The location of each level is determined as the following way. If the first level is at distance of z from the projection center, the second level would be at distance $Z_0/2$, and the nth level would be at distance $Z_0/2^{n-1}$.

One thing we must account for is that when multiple depth maps are used, there are discontinuities between consecutive levels. This can produce undersampling artifacts. This problem is overcome by interpolating between levels (Williams[98]). This is described in the next part.

Second Pass: Rendering Scenes

We use a typical Z-buffer algorithm for rendering a scene. Each surface point visible from the viewer is transformed into light source coordinates first. The illumination value at the point is then determined differently depending on which type of depth map structure is chosen.

(a) Indexing the Depth Map

If only a single depth map is generated for each light source, there is no need to search for a depth map. However, when the multiple depth map structure is used, we must use a search algorithm to find the proper depth map(s) effectively.

To find the proper depth map in a set of depth maps, we first project perspectively the given surface point p onto the depth map located at $Z_{\max}$, which in our pyramidal structure will have the highest resolution, taking the light as the projection center. Let Z_p be the depth value of p, and Z_p' be the depth value in the depth map at the projected location of p. If $Z_p \le Z_p'$, i.e. p is closer to the light, it is fully illuminated. Otherwise, Z_p' is used for searching the appropriate depth map(s).

Given Z_p', a binary search in the pre-computed Z-list is performed to find the proper depth map. If for some i ($0 \le i < n$) Z[i] = Z_p', then i is the index of depth map we need. Otherwise, we must have Z[i]<Z_p' and Z[i+1]<Z_p' for some i. Thus, i and i+1 are the indices of the two depth maps with which an interpolation will be made.

In the pyramid structure, the depth map of the first level is at distance Z_0 from the ligh source, and the ith level is at $Z_i = \frac{Z_{i-1}}{2} = \cdots = \frac{Z_0}{2^i}$. For a given Z_p', let $i = \log 2^{\frac{Z_0}{Z_p'}}$. If $\log 2^{\frac{Z_0}{Z_p'}} - i = 0$, then i is the index of the depth map we need; otherwise, $\lfloor i \rfloor$ and $\lfloor i \rfloor + 1$ are needed to do interpolation.

(b) Computing Illumination Values

First we show how to determine the illumination value with a single depth map. The interpolation method is then described. Assume the illumination value, I, is between 0.0 to 1.0, with 1.0 representing full illumination and 0.0 no illumination. A surface point at which the illumination is to be determined is first projected on the depth map perspectively as shown in Figure 4.16. If the projected point is out of the depth map boundary, then I = 1.0 since it is not obscured by any objects from the light source. Otherwise, the light source is perspectively projected onto the depth map by taking Z_p as the projection center. Let S_l be the projected area of the light source on the projection plane, and $S_{lvisible}$ be the portion of S_l

which is visible from Z_p as was computed by the Z-buffer hidden surface removal algorithm. Thus the ratio value $\frac{S_{lvisible}}{S_l}$ is the illumination value I.

The above approach is general for arbitrarily shaped light sources. However, computation costs can be improved greatly for a rectangular shaped light source, a very common light shape in the real world. Here we show a very efficient computation using the summed area table technique for rectangular light.

The summed area table technique has been applied to filtering texture in (Crow[27]). Here we demonstrate that it also can be used as a very efficient way to compute $Z_{lvisible}$. Let us temporarily replace the depth map by a shadow mask. A shadow mask is a two dimensional array each of whose elements m_{ij} is between 0.0 and 1.0 in value, where 1.0 means totally blocked, and 0.0 means not blocked. A value between 0.0 and 1.0 represents a partially blocked value. An rectangular light projected onto the mask will be a rectangle as shown in Figure 4.23. Obviously,

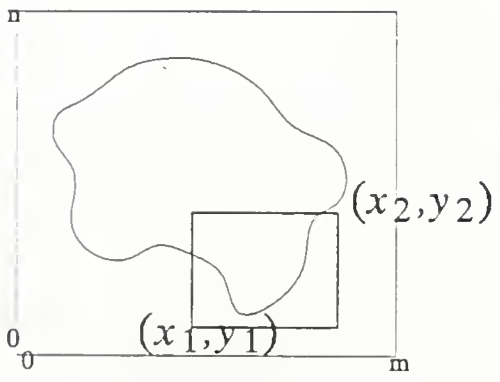


Figure 4.16 Using a summed area table to compute shades for rectangular shaped light

$$S_{lvisible} = (x_2 - x_1)(y_2 - y_1) - \sum_{\substack{x_1 \le i \le x_2 \\ y_1 \le j \le u_2}} m_{ij}$$

$$= (x_2 - x_1)(y_2 - y_1) - s_{x1,y1,x2,y2} \tag{28}$$

In the scanline method, $s_{x1,y1,x2,y2}$ is computed by a double loop for each projected rectangle. However, if we precompute a summed area table of the same dimension as the mask, each element of which is $a_{ij} = s_{0,0,i,j}$, then

$$s_{x1,y1,x2,y2} = \sum_{\substack{x_1 \le i \le x_2 \\ y_1 \le j \le u_2}} m_{ij}$$

$$= \sum_{\substack{0 \le i \le x_2 \\ 0 \le j \le y_2}} m_{ij} - \sum_{\substack{0 \le i \le x_2 \\ 0 \le j \le y_1}} m_{ij} - \sum_{\substack{0 \le i \le x_1 \\ 0 \le j \le y_2}} m_{ij} - \sum_{\substack{0 \le i \le x_1 \\ 0 \le j \le y_1}} m_{ij} \tag{29}$$

$$= a_{x_2 y_2} - a_{x_2 y_1} - a_{x_1 y_2} + a_{x_1 y_1}$$

We need to compute this summed area table only once. Then for any given rectangle with low-left corner at (x_1,y_1) and up-right at (x_2,y_2), $s_{x1,y1,x2,y2}$ can be computed by just two subtractions and one addition (Crow[27]). But (x_1,y_1) and (x_2,y_2) might not be at exact grid points. In this case, we must compute the summed area values for each of the four corners of our rectangle by bilinear interpolation. The interpolated values are then applied to (29).

We have described the shadow computation method for a single depth map. When using multiple depth map structures, suppose the optimal distance to index a depth map is at Z_p', but there is no one map in our precomputed depth map structure satisfying this exactly. Surely we would not want to use a single nearest one to do our job, since this rounding off solution could potentially generate aliasing problems in the final picture. Interpolation is again used to solve this problem. Let i and i+1 be indices of two depth maps found by the method stated previously. The distances of these two depth maps from the projection center are Z_i and Z_{i+1} respectively, and we have $Z_i < Z_p' < Z_{i+1}$. Then we first use the method for single depth maps stated above to compute the illumination value for each depth map. If we have two illumination values I_i and I_{i+1} respectively, the interpolated illumination value is then given by the formula

$$I = I_i * \frac{Z_{i+1} - Z_p'}{Z_{i+1} - Z_i} + I_{i+1} * \frac{Z_p' - Z_i}{Z_{i+1} - Z_i} \tag{30}$$

4.7.5. Experimental Results

Figures 4.24 to 4.29 illustrate some experimental results of our penumbra approximation algorithm. In this first test scene there is a tall block casting a shadow onto the floor. The light is placed at the front-left-top side of the block. The position of the light has been chosen so as to make the depth values of the different portions of the block vary over a large range as seen from the light, so that the characteristics of our algorithm can be demonstrated clearly.

Figure 4.24 shows a picture rendered by using the traditional Z-buffer algorithm, where the light is point-type. This picture gives us a good feeling about the geometrical configuration of the test scene, and also can be used to be compared with other pictures of penumbra shadows.

We now replace this point light source by a square shaped light source. Figure 4.25 to Figure 4.27 are generated by using the same light source, but with different depth map structures, which clearly show different shadow effects. Figure 4.25 (a), (b) and (c) all use a single depth map, but for each image the projection plane for computing the depth map has been placed at a different distance from the light. For Figure 4.25 (a) the projection plane of the depth map has been placed at a position near the bottom of the block. Since the shadow mask is close to the floor and far from the light, the penumbra area is the narrowest among the three pictures. For Figure 4.25 (b), the projection plane of the depth map has been placed at the middle of the block; and for Figure 4.25 (c), the projection plane of the depth map has been placed near the top of the block, which produces the fuzziest shadow of the three.

Figure 4.26 and Figure 4.27 use multiple depth map structures. Figure 4.26 is generated by using a pyramidal structure of only two levels. The values between the two levels are linearly interpolated. In this picture, the shadows of the bottom part and the top part of the block are close to the shadows of corresponding parts in Figure 4.25 (a) and Figure 4.25 (c) respectively; but the middle part is not very similar to the corresponding part of Figure 4.25 (b). Figure 4.27 is generated by using a depth map array with three levels, which shows the better penumbra shadow.

Figure 4.28 shows a synthetic ball casting a shadow penumbra on a curved natural surface. The background was digitized from a real cloth drapery by using the NYU ratio depth sensor. Figure 6.3 (see Appendix) shows the depth map of the background as a gray image. In addition to this depth image, we took an ordinary intensity image of the same scene (Figure 6.2). The original intensity values of surface points in the background were superimposed into the picture, and the depth values were used to provide the three-dimensional coordinates of the surface points for the shadow computation.

Figure 4.29 gives a comparison between our approximation method and accurate ray-tracing technique. (A) is generated by using ray-tracing technique and (b) is generated by our approximation. In these two pictures, we can see that the approximation is satisfactory in most of area, with an exception at the front-left part of the floor for the reason described in Section 4.7.3.

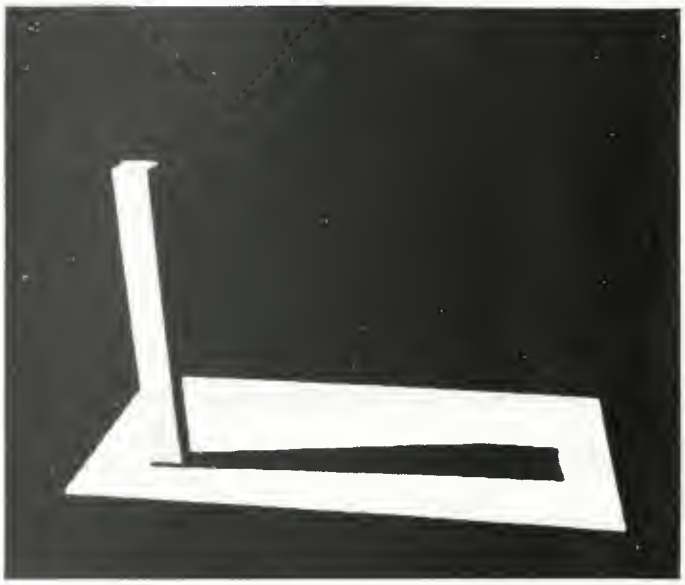
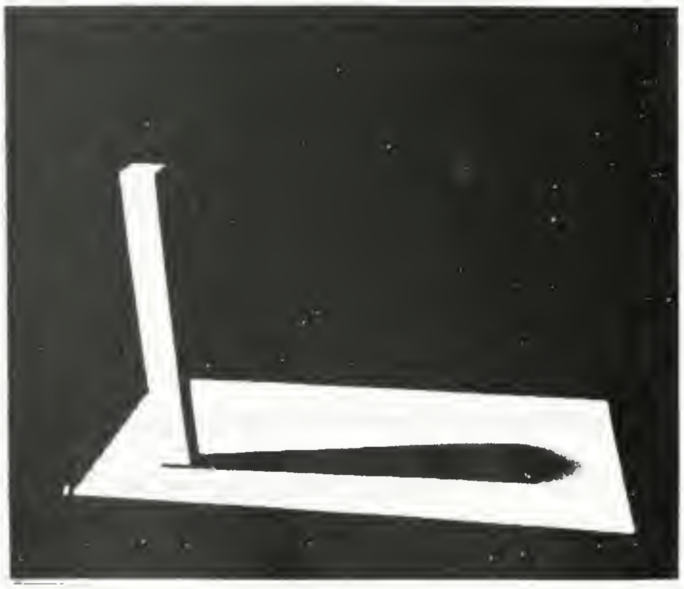
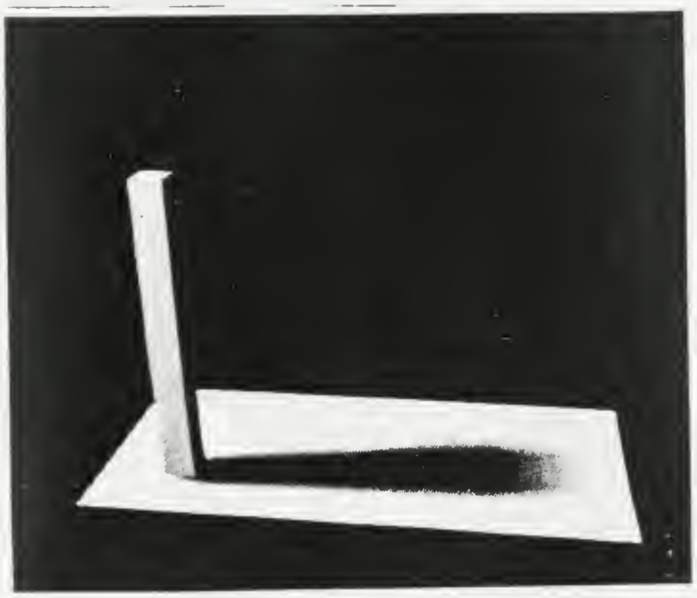


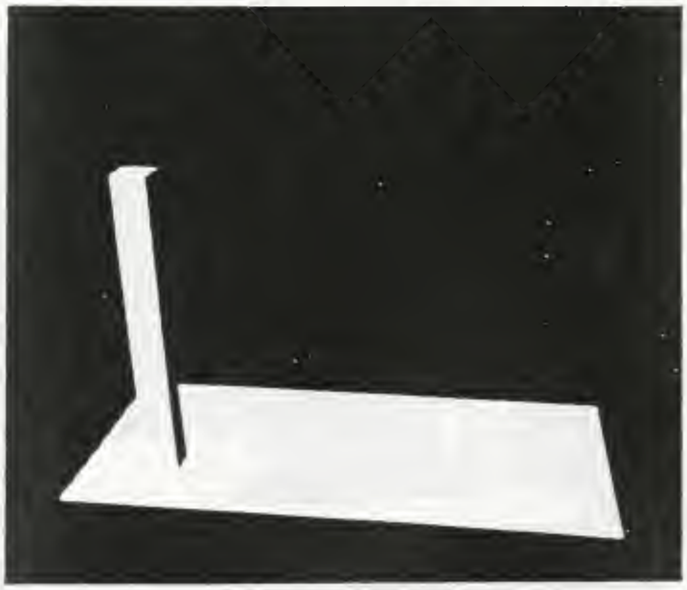
Figure 4.24



(a)



(b)



(c)
Figure 4.25

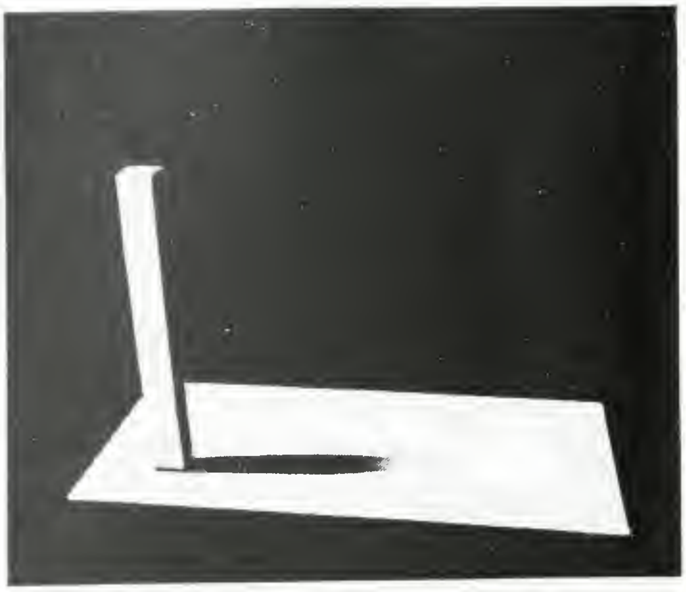


Figure 4.26

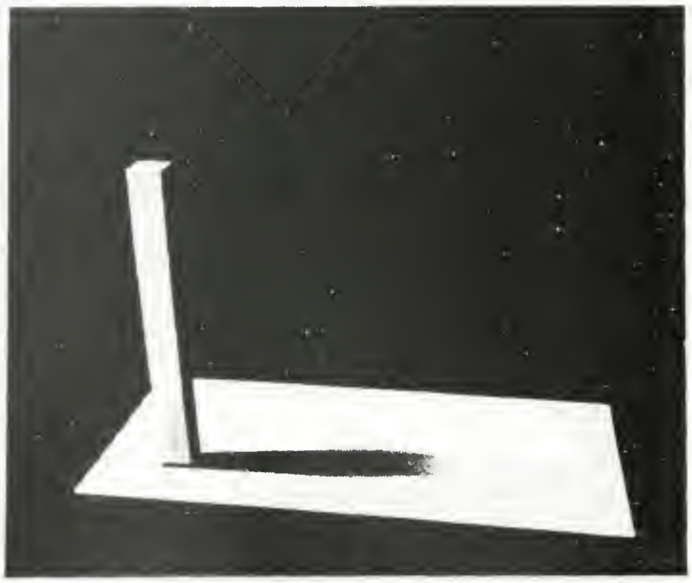


Figure 4.27

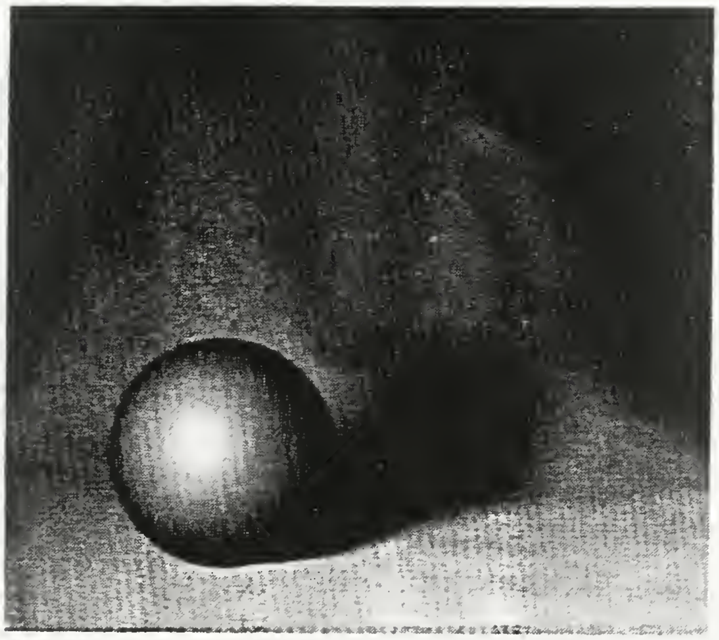
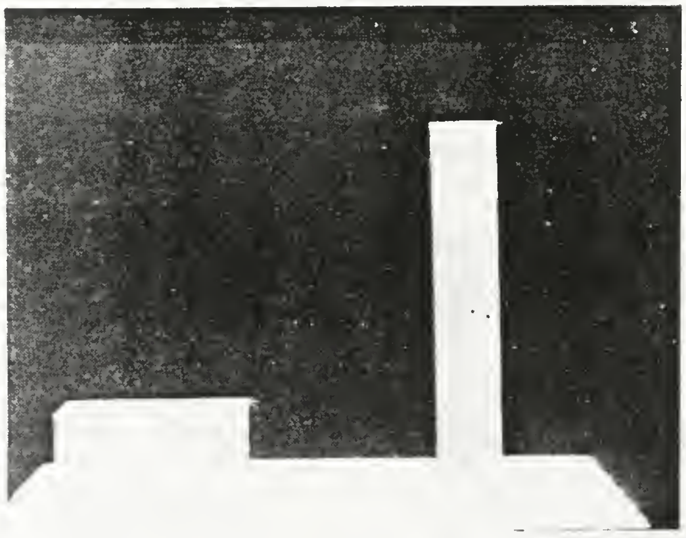
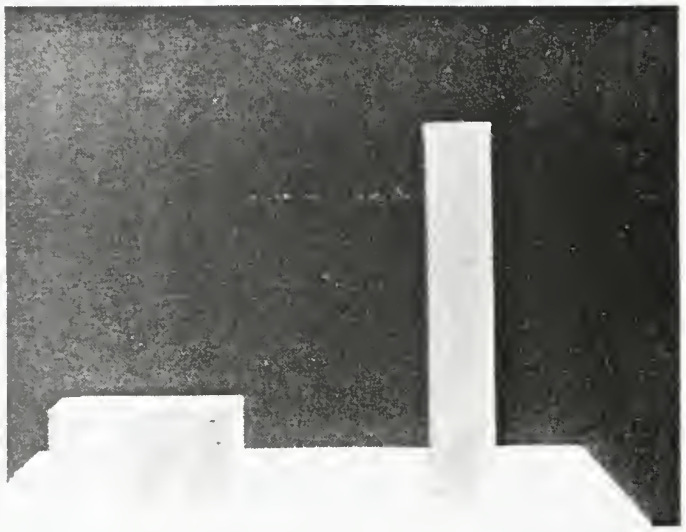


Figure 4.28



(a)



(b)
Figure 4.29

CHAPTER 5

Conclusion and Future Work

5.1. Summary

We have presented a new three-dimensional surface model - three-dimensional fuzzy disk model, for computer graphics display. It allows any curved surface to be approximated by a number of overlapping fuzzy disks covering the surface. A new blending method has been developed to generate smoothly curved surfaces from the overlapping fuzzy disks. Various of geometric properties of this blending method have been discussed. The shape of a blending surface can be intuitively controlled by varying geometric parameter.

This model uses only one primitive - a three-dimensional oriented disk. This is simpler than other existing surface primitives, such as polygons, parametric patches, etc.

Like an octree which assigns explicit sizes to primitive cubes, the primitive in our model - disk also allows explicit sizes. This property allows us to build up a multi-resolution structure analogous to the hierarchical structure of octree. Adaptive refinement is achieved with this multi-resolution structure. From the point of view of image generation, this structure supports coarse-to-fine display process.

Rendering algorithms for generating images from our model have been presented. The function used to construct a blending surface is parameterized by z depth in viewing coordinate system. A scan-line algorithm in image space is then used to construct an image. The intersection point between a viewing ray and a blending surface implicitly defined by a density function is calculated by a numerical method. The scan-line algorithm uses spatial coherence to determine an initial guess of a *z* depth for an intersection point between a viewing ray and a blending surface, as required for Newton's method, which improves the efficiency of the numerical method significantly.

The hidden-surface removal process required for image generation is improved by the subgrouping three-dimensional disks by orientations in the data structure.

We also presented a simpler, less accurate, but more efficient approximation to the original model. This approximation model will generate some error in the generated images, depending on the view point.

We also presented a shadow penumbra approximation model, capable of generating fairly realistic penumbra shadows. This model is not restricted to our three-dimensional surface model, but is general for all Z-buffer based rendering algorithms.

5.2. Future Work

(1) Anti-aliasing at occluding boundaries in our fast approximation blending method is important problem and should be studied in the future.

(2) The observations presented in Section 3.2.6 allow the shape of a blending surface to change continuously from smoothly curved shape to nearly polyhedra shape. It might be possible to establish a mathematical technique for combining smoothly curved surface and polyhedra surface into an uniform format. Furthermore, a smooth transition between two types of surfaces might be achievable by varying one or more parameters.

(3) The structure of our model can be refined to support ray-tracing and radiosity algorithms which are two important current computer graphics methods for generating realistic images.

(4) One of the properties distinguishing our model from other existing models is the regular shape of the simple primitive in our model. This property allows many of calculations in our algorithms to be implemented in simple high-speed hardware further, allowing real-time display of realistic complex three-dimensional surfaces in an inexpensive manner.

(5) Parallel implementation of our methods is obviously attractive. From the scan-line algorithm we described, one can see that the rendering of different portions of an image can obviously be performed in parallel. An implementation of our model on a general purpose parallel computer is therefore a possible approach to realize real-time dynamic display.

(6) To broaden the usefulness of our model, it is necessary to develop algorithms for constructing the data it requires from other data sources, such as CT scan data of medical imaging, besides the video-based three-dimensional sensing data as being used in this research.

(7) To improve the shadow penumbra approximation scheme that we introduced is interesting and it will make the scheme more applicable.

CHAPTER 6

Appendix: Sampling 3D Surfaces of Real Objects

6.1. Overview of 3D Sensing Techniques

There exist several different types of three-dimensional sampling techniques (*Jarvis*[47], *Boult*[19]), including use of 3D digitizers, ultrasonic depth sensors, laser time of flight depth sensors, stereo vision, triangulation methods using active lighting, etc. Here we are interested in methods which are suitable for sampling textured complex 3D surfaces of real objects at relatively high resolutions.

To be able to sample not only the three-dimensional coordinates of the points on a surface but also texture information for that surface, video-based techniques are more suitable than other techniques. Video-based techniques can also provide relatively a high resolution of sample density in an efficient way. These techniques can be categorized into two classes (*Jarvis*[47]), namely

(1) *Direct and active range finding* (*Addleman*[1], *Schwartz*[90], *Bastuscheck & Schwartz*[10], *Bolles*[18], *Chiang, et al*[21], *Jarvis*[48], and *Sato, et al*[88])

In active image acquisition, a light beam or plane is scanned over the field of view, or some kind of structured light is used to illuminate the field of view, and one or more sensors respond to this scanning or structured light and yield data for 3D image.

(2) *Passive range finding* (*Barnard*[7], *Levine*[57], *Nishihara*[69], *Nishimoto & Shirai*[70], *Quam*[79], *Williams*[99], and *Yakimovsky & Cunningham*[100])

Passive image acquisition utilizes environmental or ambient lighting to illuminate a scene.

3D sensing techniques are not a topic of this thesis. We use them only as a tool to acquire 3D data from real objects to be used in our experiments. Hence, we do not give an extensive survey in this area here. Instead, we examine only few typical systems, including NYU's Ratio Image Depth Sensor, Cyberware's ECHO System (a rotational laser scanner), and stereo vision, the first two of which have been used by us. The advantages and disadvantages of each method are described briefly.

6.1.1 Ratio Image Depth Sensor (*Schwartz*[90], *Bastuscheck & Schwartz*[10])

Figure 6.1 shows the setup of this system. An illuminating beam is projected onto a work area which is surveyed by a TV camera. The basic principle of the system is that the rays of projected light can be given some property P that is constant in each vertical plane of the beam W, but varies monotonically from left to right across the beam, which is to say monotonically with the geometric parameter h. P must be invariant under reflection, and it must be possible to sense P using a special camera. Obvious simple properties such as intensity, color, or polarization of common light cannot be used, since all of these can be changed considerably by reflection. For this reason, the simplest property that can be used for P is the ratio $R(h) = \frac{I_1(h)}{I_2(h)}$ of two intensities. Specifically, a first image is made with the scene illuminated by a beam of light, $I_1(h)$, which veries monotonically in intensity from one side to the other. Then a second image is made with a beam of uniform intensity, $I_2(h)$. If the object is in perfect focus, the ratio of reflected intensities must be equal to the ratio $R(h)$ of incident intensities. Therefore the value of $R(h)$ determines h uniquely. By knowing the displacement between the light source and the viewing camera, called *baseline*, the angle between the layer h of the lighting beam and the baseline, and the angle between the ray reflected by an object and the baseline, the depth of each point on the strip illuminated by

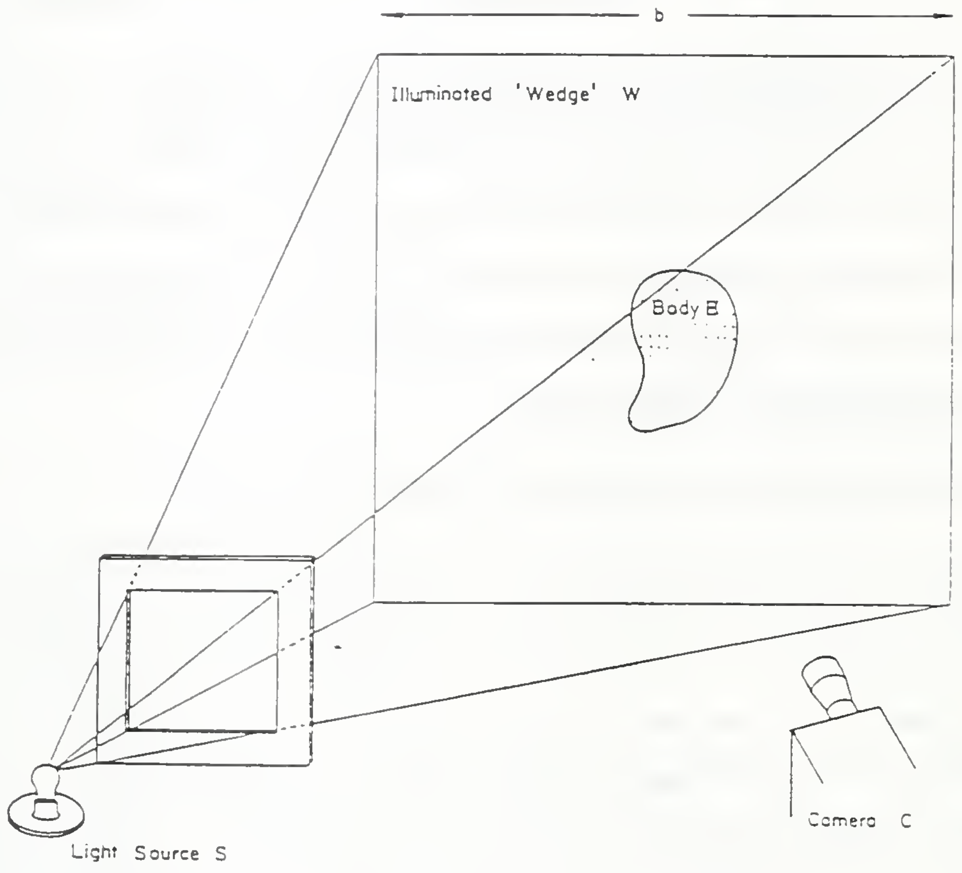


Figure 6.1 Illumination of a body by a structured wedge of light

layer *h* can be easily calculated by a triangulation method.

Figure 6.2 shows the intensity image of a piece of cloth drapery, and Figure 6.3 shows the depth map of the same cloth drapery obtained from the NYU Ratio Image Depth Sensor. The depth at each pixel position of the depth image is proportional to gray values.

There are two major restrictions in application of this scheme.

(a) *Shadows* - Since there is a displacement between the light source and the camera in this system, there may exist areas on the imaged surface which are visible from the camera but invisible from the light source. No depth information can be derived for these areas. This

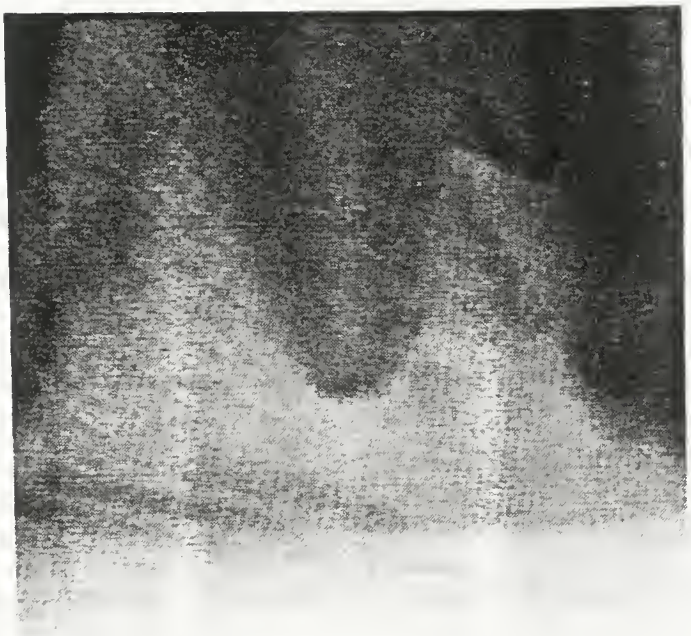


Figure 6.2 A intensity image of a cloth drapery

Figure 6.3 The depth map for the same cloth drapery of
Figure 6.2 measured by using NYU Ratio Image Depth Sensor

problem can be partially or completely resolved by introducing additional light sources placed at different positions.

(b) *Reflectivity of surfaces* - Calculation of depth at each pixel depends on the intensity of light reflected from the work scene to the camera which in turn largely depends on the reflectivity of imaged surface and the orientation of the surface. This property results in poor performance of the system on specular surfaces and surfaces which are almost tangent to the viewing direction.

6.1.2 Rotational Laser Scanner (*Addleman*[1])

Another system of interest is the ECHO system, a rotational laser scanner, developed by Cyberware Laboratory Inc, Pacific Grove, California. Figure 6.4 shows the setup of this system. An object to be digitized is placed at the center of the platform and kept stationary. An

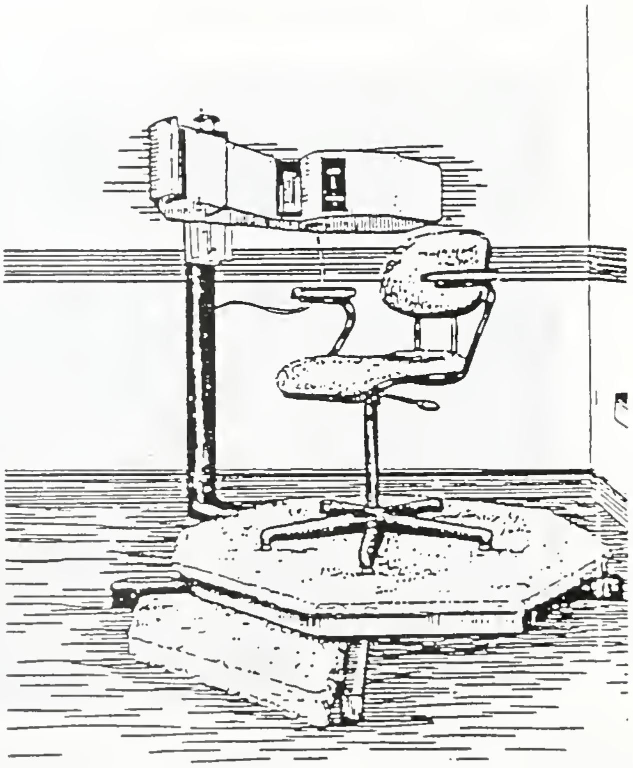


Figure 6.4 Design of a rotational laser scanner, ECHO system

arm holding the digitizer can rotate around the platform. An optical system projects a thin plane of laser light onto the object. A camera detects the illuminated strip on the objects from a certain angle away from the plane of projection. Trigonometric calculations produce the numerical radius (respect to the center of rotation) of the points on the illuminated strip. By rotating the arm, a complete radius map for the surface of the object is obtained.

Figure 3.21 illustrates an example radius map of Haydn bust which is digitized by this system. This system generates a 512×512 radius map in about 15 seconds.

The shadow problem also effects this system. In addition, a symmetric problem, called the obscured section problem, arises here. That is, from some angles, certain features of object can obscure a section of the illuminated strip. We say this problem is symmetric to shadow problem because this obscured section is visible from the light source, but invisible from the camera. This problem is partially solved by a mirror system designed for the digitizer. The camera views a strip from both sides of the plane of projection through the mirror system. Normally, the two views of strip yield duplicate measurements. But if the obscure section problem arises, the system yields only information from the curve it can see.

6.1.3 Stereo Vision

The two systems described above have a common restriction that the size of objects to be sampled should be within a certain range and the distance from an object to a sensing system should also be within a certain range because of the limited sensing space of the systems.

So-called stereo vision simulates human vision more closely, free of this objection. That is, this method is useful for sensing a scene at a large distance. Another significant advantage of this approach is that it relys solely on normal scene illumination. The basic principle of this stereo ranging method is as follows. We take n ($n \geq 2$) images from different viewpoints, find points that correspond in at least two images, and use trigonometry to calculate depth. The critical and also the most difficult step in the stereo method is to find corresponding points in two images. This is sometimes called the 'matching' or 'disparity' problem.

The most common approach to solving the corresponding point problem is often a correlation algorithm. In such an algorithm, a correlation operator is applied to a pair of images to find the disparities for all corresponding points between the two images. Various operators can be used for this (*Levine*[57], *Yakimovsky & Cunningham*[100]). Features such as zero-crossing points are sometimes used to improve the matching process (*Nishimoto & Shirai*[70]). Nishimoto and Shirai's method finds disparities progressively from coarse level to fine level by using Laplacian-Gaussian filtered pyramidal images. Disparities calculated at coarser level is used to effectively guide the calculations of disparities at finer levels. *Marr and Poggio*[63] present an elegant relaxation algorithm for finding disparities.

Most natural scenes have complicated irregular features which makes the correlation algorithm particularly successful, while such an algorithm often fails with featureless scenes or scenes with cyclic patterns. For example, *PIXAR*[77] demonstrated a depth map of a valley recovered from a pair of intensity images of the valley from different view points by using stereo method. This depth map provides the three-dimensional shape information of the valley which is then used together with intensity image for computer graphics applications.

6.2. Preprocessing of Raw 3D Sample data

Because of the limited accuracy of available 3D sensing techniques, there is always a certain amount of noise contained in the raw sample data. Some pre-processing of the raw sample data is generally required to make it usable for computer graphics applications.

In general, it is not possible to tell whether some high frequency components in raw sample data are noise or actual surface detail. So it is difficult (or may not be possible) to develop algorithms to do automatic noise filtering since it might remove actual surface details.

Because of the above, we believe a reasonable resolution to this problem is to provide graphics application developers with an interactive tool so they can do selective filtering for raw sample data. We developed an interactive environment for this purpose.

6.3. Extracting Texture from the Sampled Surface

The color intensity value measured through a video camera is the reflected intensity from a surface. Surface reflectance varies from one material to another, and this is part of what is detected. However, the intensity value measured through a video camera is affected by many other factors, such as spectra and intensity of light sources, cameras, etc. We aim to recover the surface reflectance; this is a constant color property of surfaces which is invariant under different environments. *Maloney*[62] presents an algorithm for estimating the surface reflectance functions in a scene with incomplete knowledge of the spectral power distribution of ambient light, but under certain restrictions on the range of lights and surfaces that the visual system will encounter. We have used a much simpler method to gather surface color intensity through a video camera which provides satisfactory results for computer graphics applications.

This simple approach is just to use ambient light to illuminate the surfaces which we want to sample. But since it is difficult to get adequate ambient light sources, shading effects in very low spatial frequencies tend to be problematic. Also in some natural scene, the spectral composition of the ambient light varies with spatial locations (*Maloney*[62]). For these reasons, we need to estimate and discount a slowly varying (spatial-low-pass) ambient light and shading. High-pass filters which have been well studied in image processing are used in our experiments to the texture extraction process.

REFERENCES

[1] D.Addleman and L.Addleman, *Rapid 3D Digitizing*, Computer Graphics World, pp.41-44, Nov. 1985.

[2] J. Amanatides, *Ray Tracing with Cones*, Computer Graphics, Vol.18, No.3, July 1984, pp.129-136.

[3] J. Amanatides, *Realism in Computer Graphics: A Survey*, IEEE Computer Graphics and Applications, Vol.7, No.1, January 1987, pp.44-56.

[4] J. Arvo and D. Kirk, *Fast Ray Tracing by Ray Classification*, Computer Graphics, Vol.21, No.4, July 1987, pp.55-64.

[5] P.R. Atherton, *A Scan-Line Hidden Surface Removal Procedure for Constructive Sold Geometry*, Computer Graphics, Vol.17, No.3, July 1983, pp.73-82.

[6] D.H. Ballard and C.M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[7] S.T. Barnard, *A Stochastic Approach to Stereo Vision*, Reading in Computer Vision: Issues, Problems, Principles, and Paradigms, ed. M.A. Fischler and O. Firschein, Morgan Kaufmann, Los Altos, CA, 1987, pp.21-25.

[8] A.H. Barr, *Ray Tracing Deformed Surfaces*, Computer Graphics, Vol.20, No.4, August 1986, pp.287-296.

[9] B.A. Barsky, *Computer Graphics and Geometric Modeling Using Beta-splines*, Springer-Verlag, Heidelberg, 1987.

[10] C.M.Bastuscheck and J.T.Schwartz, *Experimental Implementation of a Ratio Image Depth Sensor*, Techniques for 3D Machine Perception, ed. A.Rosenfeld, pp.1-12, Elsevier Science Publishers B.V. (North-Holland), 1986.

[11] B.G. Baumgart, *A Polyhedron Representation for Computer Vision*, NCC, 1975, pp.589-596.

[12] L.Bergman, H.Fuchs and E.Grant, *Image Rendering by Adaptive Refinement*, Computer Graphics, Vol.20, No.4, August 1986, pp.29-37.

[13] P. Bezier, *Mathematical and Practical Possibilities of UNISURF*, Computer Aided Geometric Design, ed. R.E. Barnhill and R.F. Riesenfeld, Academic, New York, 1974.

[14] P. Bezier, *Numerical Control - Mathematics and Applications*, A.R. Forrest (trans.) Wiley, London, 1972.

[15] J.F. Blinn, *A Generalization of Algebraic Surface Drawing*, ACM Transactions on Graphics, Vol.1, No.3, July 1982, pp.235-256.

[16] J.F.Blinn, *Computer Display of Curved Surfaces*, Thesis, Computer Science Dept., U. of Utah, Salt Lake City, Utah, 1978.

[17] J.F. Blinn, *Models of Lights Reflection for Computer Synthesized Pictures*, Computer Graphics, Vol.11, No.2, July 1977.

[18] R.C. Bolles, J.H. Kremers and R.A. Cain, *A Simple Sensor to Gather Three-Dimensional Data*, SRI International, Tech. Note 249, 1981.

[19] T.E. Boult, *A Survey of Some Three Dimensional Vision Systems*, SIGART Newsletter, No.92, April 1985, pp.28-37.

[20] E.E. Catmull, *Computer Display of Curved Surfaces*, Proc. IEEE Conf. Computer Graphics, Pattern Recognition and Data Structures, Los Angeles, Calif., May 1975.

[21] M.C. Chiang, J.B.K. Tio and E.L. Hall, *Robot Vision Using a Projection Method*, Proc. 3rd ICRVSC, ed. B. Rocks, 1983, pp.113-120.

[22] M.F. Cohen, D.P. Greenberg, D.S. Immel and P.J. Brock, *An Efficient Radiosity Approach for Realistic Image Synthesis*, IEEE Computer Graphics and Applications, Vol.6, No.3, March 1986.

[23] S.D. Conte and Carl de Boor, *Elementary Numerical Analysis - An Algorithmic Approach*, 3rd ed., McGraw-Hill, NY, 1980.

[24] R.L. Cook, T. Porter and L. Carpenter, *Distributed Ray Tracing*, Computer Graphics, Vol.18, No.3, July 1984, pp.137-146.

[25] S.A. Coons, *Surfaces for Computer Aided Design of Space Forms*, MIT Project Mac, TR-41, June 1967.

[26] F.C. Crow, *Shadow Algorithms for Computer Graphics*, Computer Graphics, Vol.11, No.2, Summer 1977, pp.442-448.

[27] F.C. Crow, *Summed-Area Tables for Texture Mapping*, Computer Graphics, Vol.18, No.3, July 1984, pp.207-212.

[28] L.Doctor and J.G.Torborg, *Display Techniques for Octree-Encoded Objects*, IEEE Computer Graphics & Applications, Vol. 1, No. 3, pp.29-38, 1981.

[29] I.D. Faux and M.J. Pratt, *Computational Geometry for Design and Manufacture*, John Wiley, New York, 1979.

[30] E.A. Feibush, M. Levoy and R.L. Cook, *Synthetic Texturing Using Digital Filters*, Computer Graphics, Vol.14, No.3, July 1980, pp.294-301.

[31] J. Ferguson, *Multivariate Curve Interpolation*, Journal of he ACM, Vol.11, No.2, 1964, pp.461-482.

[32] J.D.Foley and A.Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Co., 1982.

[33] A. Fournier, D. Fussell and L. Carpenter, *Computer Rendering of Stochastic Models*, Communications of the ACM, Vol.25, No.6, June 1982.

[34] G.Frieder, D.Gordon and R.A.Reynolds, *Back-to-Front Display of Voxel Based Objects*, IEEE Computer Graphics & Applications, Vol. 5, No. 1, pp.52-60, Jan, 1985.

[35] H. Fuchs, G.D. Abram, and E.D. Grant, *Near Real-Time Shaded Display of Rigid Objects*, Computer Graphics, Vol.17, No.3, July 1983, pp.65-72.

[36] I.Gargantini, *Linear Octrees for Fast Processing of Three-Dimensional Objects*, Computer Graphics and Image Processing 20, pp.365-374, 1982.

[37] I.Gargantini, T.R.Walsh and O.L.Wu, *Viewing Transformations of Voxel-Based Objects via Linear Octrees*, IEEE Computer Graphics and Applications, pp.12-21, Oct. 1986.

[38] A.S. Glassner, *Space Subdivision for Fast Ray Tracing*, IEEE Computer Graphics and Applications, Vol.4, No.10, Oct. 1984, pp.15-22.

[39] D. Gordon, and R.A. Reynolds, *Image Space Shading of 3-Dimensional Objects*, Computer Graphics and Image Processing, Vol.29, No.3, March 1985, pp.361-376.

[40] H.Gouraud, *Continuous Shading of Curved Surfaces*, IEEE Trans. Computers, C-20, June 1971.

[41] P.S. Heckbert and P. Hanrahan, *Beam Tracing Polygonal Objects*, Computer Graphics, Vol.18, No.3, July 1984, pp.119-128.

[42] P.S. Heckbert, *Survey of Texture Mapping*, IEEE Computer Graphics and Applications, Vol.6, No.11, Nov. 1986, pp.56-67.

[43] G.T. Herman and H.K. Liu, *Tree-Dimensional Display of Human Organs from Computer Tomograms*, Computer Graphics and Image Processing, Jan. 1979, pp.1-21.

[44] C. Hoffmann and J. Hopcroft, *Quadratic Blending Surfaces*, Computer-Aided Design, Vol.18, No.6, July/August 1986, pp.301-306.

[45] T.V. Hook, *Real-Time Shaded NC Milling Display*, Computer Graphics, Vol.20, No.4, August 1986, pp.15-20.

[46] J.E. Hopcroft, *The Impact of Robotics on Computer Science*, Communications of the ACM, Vol.29, No.6, June 1986, pp.486-498.

[47] R.A. Jarvis, *A Perspective on Range Finding Techniques for Computer Vision*, IEEE Tran. on Pattern Analysis and Machine Intelligence, Vol. PAMI-5, No.2, March 1983, pp.122-139.

[48] R.A. Jarvis, *A Laser Time-of-Flight Range Scanner for Robotic Vision*, Australian Nat. Univ., Computer Science Tech. Rep. TR-CS-81-10.

[49] K.I. Joy and M.N. Bhetanabhotla, *Ray Tracing Parametric Surface Patches Utilizing Numerical Techniques and Ray Coherence*, Computer Graphics, Vol.20, No.4, August 1986, pp.279-285.

[50] J.T. Kajiya, *The Rendering Equation*, Computer Graphics, Vol.20, No.4, August 1986, pp.143-150.

[51] M.R. Kaplan, *The Use of Spatial Coherence in Ray Tracing*, Techniques for Computer Graphics, ed. D. Rogers and R. Earnshae, Springer-Verlag, New York 1987.

[52] T.L. Kay and J.T. Kajiya, *Ray Tracing Complex Scenes*, Computer Graphics, Vol.20, No.4, August 1986, pp.269-278.

[53] E. Keppel, *Approximating Complex Surfaces by Triangulation of Contour Lines*, IBM J. Res. Develop Vol.19, No.1, Jan. 1975, pp.2-11.

[54] E. Kishon and X.D. Yang, *A Video Camera Interface for High-Speed Region Boundary Location*, New York Univ., Courant Institute, Tech. Rep. 157, 1985.

[55] D.H. Laidlaw, W.B. Trumbore and J.F. Hughes, *Constructive Solid Geometry for Polyhedral Objects*, Computer Graphics, Vol. 20, No. 4, August 1986, pp.161-170.

[56] J.M.Lane, L.C.Carpenter, T.Whitted and J.F.Blinn, *Scan Line Methods for Displaying Parametrically Defined Surfaces*, Communications of ACM, Vol. 23, No. 1, pp.468-479, Jan. 1980.

[57] M.D. Levine, *Computer Determination of Depth Maps*, CGIP, Vol.2, 1973, pp.131-150.

[58] M. Levoy, *Display of Surfaces from Volume Data*, IEEE Computer Graphics and Applications, Vol.8, No.3, May 1988.

[59] R.A. Lewis and A.R. Johnston, *A Scanning Laser Range Finder for a Robotic Vehicle*, Proc. 5th Int. Joint Conf. Artificial Intelligence, 1977, pp.762-768.

[60] H.K. Liu, *Two- and Three-Dimensional Boundary Detection*, Computer Graphics and Image Processing 6, pp.123-134, 1977.

[61] W.E. Lorensen and H.E. Cline, *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*, Computer Graphics, Vol.21, No.4, July 1987, pp.163-169.

[62] L.T. Maloney and B.A. Wandell, *Color Constancy: a Method for Recovering Surface Spectral Reflectance*, Reading in Computer Vision - Issues, Problems, Principles, and Paradigms, ed. M.A. Fischler, and O. Firschein, Morgan Kaufmann, Los Altos, CA, 1987pp.293-297.

[63] D. Marr and T. Poggio, *Cooperative Computation of Stereo Disparity*, Science 194, 1976, pp.283-287.

[64] N.L. Max, *Atmospheric Illumination and Shadows*, Computer Graphics, Vol.20, No.4, August 1986, pp.117-124.

[65] D. Meagher, *Geometric Modeling Using Octree Encoding*, Computer Graphics and Image Processing 19, pp.129-147, 1982.

[66] A. Middleditch and K. Sears, *Blend Surfaces for Set Theoretic Volume Modeling Systems*, Computer Graphics, Vol.19, No.3, July 1985, pp.161-170.

[67] G.S.P. Miller, *The Definition and Rendering of Terrain Maps*, Computer Graphics, Vol.20, No.4, August 1986, pp.39-44.

[68] E. Nakamae, K. Harada, T. Ishizaki and T. Nishita, *A Montage Method: The Overlaying of the Computer Generated Images onto a Background Photograph*, Computer Graphics, Vol.20, No.4, August 1986, pp.207-214.

[69] H.K. Nishihara, *Practical Real-Time Imaging Stereo Matcher*, Optical Engineering Vol.23, No.5, Sept./Oct. 1984, pp.536-545

[70] Y. Nishimoto and Y. Shirai, *A Parallel Matching Algorithm for Stereo Vision*, Proc. 9th IJCAI, Vol.2 1985, pp.977-980.

[71] T. Nishita and E. Nakamae, *Continuous Tone Representation of Three-dimensional Objects Taking Account of Shadows and Interreflection*, Computer Graphics, Vol.19, No.3, July 1985, pp.23-30.

[72] T. Pavlidis, *Algorithms for Computer Graphics and Image Processing*, Computer Science Press, Rockville, MD, 1982.

[73] K. Perlin, *An Image Synthesizer*, Computer Graphics, Vol.19, No.3, July 1985.

[74] K. Perlin, *Synthesizing Realistic Textures by the Composition of Perceptually Motivated Functions*, Ph.D. thesis, Computer Science Dept., New York Univ., 1986.

[75] K. Perlin, *The Noise Function*, Note, R/Greenberg Associates and New York University, 1986.

[76] B-T.Phong, *Illumination for Computer Generated Pictures*, Comm. ACM 18, 6, June 1975.

[77] PIXAR, *Exhibition at SIGGRAPH'87 Conference*, Anaheim, CA, July 27-31, 1987.

[78] M. Potmesil, *Generating Models of Solid Objects by Matching 3D Surface Segments*, 8th IJCAI, 1983, pp.1089-1093.

[79] L.H. Quam, *Hierarchical Warp Stereo*, Readings in Computer Vision: Issues, Problems, Principles, and Paradigms, ed. M.A. Fischler and O. Firschein, Morgan Kaufmann, Los Altos, CA, 1987, pp.80-86.

[80] P.Quarendon and J.R.Woodwark, *The Model for Graphics*, Techniques for Computer Graphics, ed. D.F.Rogers, and R.A.Earnshaw, pp.39-63, Springer-Verlag, 1987

[81] W.T.Reeves, *Particle Systems - A Technique for Modeling a Class of Fuzzy Objects*, ACM Transactions on Graphics, pp.91-108, Vol. 2, No. 2, April 1983.

[82] W.T. Reeves, D. Salesin and R.L. Cook, *Rendering Antialised Shadows with Depth Maps*, Computer Graphics, Vol.21, No.4, July 1987, pp.283-291.

[83] A. Rockwood and J. Owen, *Blending Surfaces in Solid Geometric Modeling*, Proc. SIAM Conf. on Geometric Modeling and Robotics, Albany, NY, July 1985.

[84] D.F. Rogers and J. Alan Adams, *Mathematical Elements for Computer Graphics*, McGraw-Hill, NY, 1976.

[85] J. Rossignac, *Blending and Offsetting Solid Models*, Ph.D. Thesis, Dept. EE, Univ. of Rochester, Rochester, NY, July 1985.

[86] J. Rossignac and A. Requicha, *Constant-radius Blending in Solid Modeling*, Comput. Mech. Eng., July 1984, pp.65-73.

[87] S.M. Rubin and T. Whitted, *A Three-Dimensional Representation for Fast Rendering of Complex Scenes*, Computer Graphics, Vol.17, No.3, July 1983, pp.91-102.

[88] Y. Sato, H. Kitagawa and H. Fujita, *Shape Measurement of Curved Objects Using Multiple Slit-ray Projections*, IEEE Tran. on PAMI-4, No.6, 1982, pp.641-646.

[89] F.J.M. Schmitt, B.A.Barsky and W.Du, *An Adaptive Subdivision Method for Surface-Fitting from Sampled Data*, Computer Graphics, Vol.20, No.4, August 1986, pp.179-185.

[90] J.T. Schwartz, *Structured Light Sensors for 3-D Robot Vision*, New York Univ., Courant Institute, Tech. Rep. 65, 1983.

[91] M. Shinya, T. Takahashi and S. Naito, *Principles and Applications of Pencil Tracing*, Computer Graphics, July 1987, pp.45-54.

[92] I.E. Sutherland, R.F. Sproull and R.A. Schumacker, *A Characterization of Ten Hidden-Surface Algorithms*, Computing Surveys, Vol.6, No.1, March 1974, pp.387-441.

[93] R.W. Swanson and L.J. Thayer, *A fast Shaded-Polygon Renderer*, Computer Graphics, Vol.20, No.4, August 1986, pp.95-101.

[94] J. Weil, *The Synthesis of Cloth Objects*, Computer Graphics, Vol.20, No.4, August 1986, pp.49-53.

[95] J.Weng and N.Ahuja, *Octrees of Objects in Arbitrary Motion: Representation and Efficiency*, Computer Vision, Graphics, and Image Processing 39, pp.167-185, 1987.

[96] J.T.Whitted, *A Scan Line Algorithm for Computer Display of Curved Surfaces*, SIGGRAPH'78 Conf. Proc., Atlanta, GA, 1978.

[97] L. Williams, *Casting Curved Shadows on Curved Surfaces*, SIGGRAPH'78 Conference Proceedings, pp.270-274.

[98] L. Williams, *Pyramidal Parametrics*, Computer Graphics, Vol.17, No.3, July 1983, pp.1-11.

[99] T.D. Williams, *Depth from Camera Motion in a Real World Scene*, IEEE Tran. on PAMI-2, No.6, 1980, pp.511-516.

[100] Y. Yakimovsky and R. Cunningham, *A System for Extracting Three-Dimensional Measurements from a Stereo Pair of TV Cameras*, CGIP, Vol.7, 1978, pp.195-210.

NYU COMPSCI TR-414
Yang, Xue Dong
Fuzzy disk modeling and
rendering of textured
complex 3D... c.1

DATE DUE	BORROWER'S NAME